Contents

About This Book ......................................................................................................................... xix

CHAPTER 1  File Locations and Installation Settings .................................  1
  Installation directory structure ................................................................. 2
  How Sybase IQ locates files ................................................................. 3
    Simple file searching ................................................................. 4
    Extensive file searching............................................................. 5
  Environment variables................................................................. 5
    Setting environment variables ................................................... 6
    IQDIR15 environment variable .................................................. 7
    IQPORT environment variable .................................................. 8
    IQLOGDIR15 environment variable .......................................... 8
    IQTIMEOUT environment variable ............................................ 9
    IQTMP15 environment variable................................................. 9
    LIBPATH or LD_LIBRARY_PATH environment variable ........ 10
    PATH environment variable .................................................... 11
    SACHARSET environment variable ........................................ 11
    SALANG environment variable ............................................... 11
    SQLCONNECT environment variable ..................................... 12
    SYBASE environment variable................................................ 12
    $SYBASE_JRE6_32, $SYBASE_JRE6_64, $SYBASE_JRE5_64
      environment variables ...................................................... 13
    SYBASE_OCS environment variable........................................ 13
  Registry entries .............................................................................. 14
    Current user and local machine settings .................................. 14
    Registry structure .................................................................... 14
    Registry settings on installation............................................... 15

CHAPTER 2  SQL Language Elements ......................................................  17
  Keywords ........................................................................................ 17
  Reserved words ...................................................................... 18
  Identifiers ............................................................................... 21
  Strings ..................................................................................... 22
## Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expressions</td>
<td>23</td>
</tr>
<tr>
<td>Constants in expressions</td>
<td>25</td>
</tr>
<tr>
<td>Column names in expressions</td>
<td>25</td>
</tr>
<tr>
<td>Subqueries in expressions</td>
<td>25</td>
</tr>
<tr>
<td>SQL operators</td>
<td>26</td>
</tr>
<tr>
<td>IF expressions</td>
<td>29</td>
</tr>
<tr>
<td>CASE expressions</td>
<td>29</td>
</tr>
<tr>
<td>Compatibility of expressions</td>
<td>31</td>
</tr>
<tr>
<td>Search conditions</td>
<td>33</td>
</tr>
<tr>
<td>Comparison conditions</td>
<td>34</td>
</tr>
<tr>
<td>Subqueries in search conditions</td>
<td>35</td>
</tr>
<tr>
<td>ALL or ANY conditions</td>
<td>39</td>
</tr>
<tr>
<td>BETWEEN conditions</td>
<td>40</td>
</tr>
<tr>
<td>LIKE conditions</td>
<td>40</td>
</tr>
<tr>
<td>IN conditions</td>
<td>43</td>
</tr>
<tr>
<td>CONTAINS conditions</td>
<td>43</td>
</tr>
<tr>
<td>EXISTS conditions</td>
<td>44</td>
</tr>
<tr>
<td>IS NULL conditions</td>
<td>44</td>
</tr>
<tr>
<td>Conditions with logical operators</td>
<td>45</td>
</tr>
<tr>
<td>NOT conditions</td>
<td>45</td>
</tr>
<tr>
<td>Truth value conditions</td>
<td>45</td>
</tr>
<tr>
<td>Three-valued logic</td>
<td>46</td>
</tr>
<tr>
<td>User-supplied condition hints</td>
<td>46</td>
</tr>
<tr>
<td>Special values</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT DATABASE special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT DATE special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT PUBLISHER special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT TIME special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT TIMESTAMP special value</td>
<td>54</td>
</tr>
<tr>
<td>CURRENT USER special value</td>
<td>54</td>
</tr>
<tr>
<td>LAST USER special value</td>
<td>54</td>
</tr>
<tr>
<td>SQLCODE special value</td>
<td>55</td>
</tr>
<tr>
<td>SQLSTATE special value</td>
<td>55</td>
</tr>
<tr>
<td>TIMESTAMP special value</td>
<td>56</td>
</tr>
<tr>
<td>USER special value</td>
<td>56</td>
</tr>
<tr>
<td>Variables</td>
<td>56</td>
</tr>
<tr>
<td>Local variables</td>
<td>57</td>
</tr>
<tr>
<td>Connection-level variables</td>
<td>59</td>
</tr>
<tr>
<td>Global variables</td>
<td>59</td>
</tr>
<tr>
<td>Comments</td>
<td>65</td>
</tr>
<tr>
<td>NULL value</td>
<td>67</td>
</tr>
</tbody>
</table>

### CHAPTER 3  SQL Data Types ................................................................. 69

Character data types ................................................................. 69
Contents

Numeric data types ................................................................. 73
Binary data types ................................................................. 77
Bit data type ........................................................................ 82
Date and time data types ......................................................... 82
Sending dates and times to the database .............................. 84
Retrieving dates and times from the database ....................... 85
Comparing dates and times ..................................................... 85
Using unambiguous dates and times ....................................... 86
Domains ................................................................................ 87
Data type conversions ............................................................ 89
Sybase IQ binary load format .................................................. 94

CHAPTER 4 SQL Functions .................................................. 101
Overview ................................................................................ 102
Aggregate functions ............................................................... 102
Analytical functions ............................................................... 104
Data type conversion functions .............................................. 108
Date and time functions ........................................................ 109
  Date parts ........................................................................ 112
HTTP functions ..................................................................... 114
Numeric functions ................................................................ 114
String functions .................................................................. 115
System functions .................................................................. 118
  Connection properties ....................................................... 121
  Properties available for the server .................................... 121
  Properties available for each database ................................. 122
SQL and Java user-defined functions ..................................... 122
Time series and forecasting functions .................................... 123
  Loading the IMSL libraries for time series and forecasting functions ... 124
Miscellaneous functions ........................................................ 127
Alphabetical list of functions .................................................. 128
  ABS function [Numeric] .................................................... 128
  ACOS function [Numeric] .................................................. 128
  ARGN function [Miscellaneous] ......................................... 129
  ASCII function [String] ..................................................... 129
  ASIN function [Numeric] ................................................... 129
  ATAN function [Numeric] .................................................. 130
  ATAN2 function [Numeric] ............................................... 130
  AVG function [Aggregate] ................................................. 131
  BIGNUMTOHEX function [Data type conversion] ............... 132
  BIT_LENGTH function [String] ......................................... 132
  BYTE_LENGTH function [String] ....................................... 133
  CAST function [Data type conversion] ............................... 133

Reference: Building Blocks, Tables, and Procedures
<table>
<thead>
<tr>
<th>Function</th>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEIL function</td>
<td>Numeric</td>
<td>134</td>
</tr>
<tr>
<td>CEILING function</td>
<td>Numeric</td>
<td>135</td>
</tr>
<tr>
<td>CHAR function</td>
<td>String</td>
<td>135</td>
</tr>
<tr>
<td>CHAR_LENGTH function</td>
<td>String</td>
<td>136</td>
</tr>
<tr>
<td>CHARINDEX function</td>
<td>String</td>
<td>136</td>
</tr>
<tr>
<td>COALESCE function</td>
<td>Miscellaneous</td>
<td>137</td>
</tr>
<tr>
<td>COL_LENGTH function</td>
<td>System</td>
<td>138</td>
</tr>
<tr>
<td>COL_NAME function</td>
<td>System</td>
<td>138</td>
</tr>
<tr>
<td>CONNECTIONPROPERTY function</td>
<td>System</td>
<td>139</td>
</tr>
<tr>
<td>CONVERT function</td>
<td>Data type conversion</td>
<td>139</td>
</tr>
<tr>
<td>CORR function</td>
<td>Aggregate</td>
<td>142</td>
</tr>
<tr>
<td>COS function</td>
<td>Numeric</td>
<td>143</td>
</tr>
<tr>
<td>COT function</td>
<td>Numeric</td>
<td>143</td>
</tr>
<tr>
<td>COVAR_POP function</td>
<td>Aggregate</td>
<td>144</td>
</tr>
<tr>
<td>COVAR_SAMP function</td>
<td>Aggregate</td>
<td>145</td>
</tr>
<tr>
<td>COUNT function</td>
<td>Aggregate</td>
<td>146</td>
</tr>
<tr>
<td>CUME_DIST function</td>
<td>Ranking</td>
<td>147</td>
</tr>
<tr>
<td>DATALength function</td>
<td>System</td>
<td>148</td>
</tr>
<tr>
<td>DATE function</td>
<td>Date and time</td>
<td>149</td>
</tr>
<tr>
<td>DATEADD function</td>
<td>Date and time</td>
<td>149</td>
</tr>
<tr>
<td>DATECEILING function</td>
<td>Date and time</td>
<td>150</td>
</tr>
<tr>
<td>DATEDIFF function</td>
<td>Date and time</td>
<td>151</td>
</tr>
<tr>
<td>DATEDIFF function</td>
<td>Date and time</td>
<td>151</td>
</tr>
<tr>
<td>DATEFORMAT function</td>
<td>Date and time</td>
<td>155</td>
</tr>
<tr>
<td>DATENAME function</td>
<td>Date and time</td>
<td>156</td>
</tr>
<tr>
<td>DATEPART function</td>
<td>Date and time</td>
<td>157</td>
</tr>
<tr>
<td>DATEROUND function</td>
<td>Date and time</td>
<td>157</td>
</tr>
<tr>
<td>DATETIME function</td>
<td>Date and time</td>
<td>159</td>
</tr>
<tr>
<td>DAY function</td>
<td>Date and time</td>
<td>159</td>
</tr>
<tr>
<td>DAYNAME function</td>
<td>Date and time</td>
<td>160</td>
</tr>
<tr>
<td>DAYS function</td>
<td>Date and time</td>
<td>160</td>
</tr>
<tr>
<td>DB_ID function</td>
<td>System</td>
<td>161</td>
</tr>
<tr>
<td>DB_NAME function</td>
<td>System</td>
<td>162</td>
</tr>
<tr>
<td>DB_PROPERTY function</td>
<td>System</td>
<td>162</td>
</tr>
<tr>
<td>DEGREES function</td>
<td>Numeric</td>
<td>163</td>
</tr>
<tr>
<td>DENSE_RANK function</td>
<td>Analytical</td>
<td>163</td>
</tr>
<tr>
<td>DIFFERENCE function</td>
<td>String</td>
<td>165</td>
</tr>
<tr>
<td>DOW function</td>
<td>Date and time</td>
<td>165</td>
</tr>
<tr>
<td>ERRORMSG function</td>
<td>Miscellaneous</td>
<td>166</td>
</tr>
<tr>
<td>EVENT_CONDITION function</td>
<td>System</td>
<td>166</td>
</tr>
<tr>
<td>EVENT_CONDITION_NAME function</td>
<td>System</td>
<td>168</td>
</tr>
<tr>
<td>EVENT_PARAMETER function</td>
<td>System</td>
<td>168</td>
</tr>
<tr>
<td>EXP function</td>
<td>Numeric</td>
<td>169</td>
</tr>
<tr>
<td>EXP_WEIGHTED_AVG function</td>
<td>Aggregate</td>
<td>170</td>
</tr>
<tr>
<td>Function</td>
<td>Category</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>---------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>NEWID function</td>
<td>Miscellaneous</td>
<td>208</td>
</tr>
<tr>
<td>NEXT_CONNECTION function</td>
<td>System</td>
<td>209</td>
</tr>
<tr>
<td>NEXT_DATABASE function</td>
<td>System</td>
<td>209</td>
</tr>
<tr>
<td>NEXT_HTTP_HEADER function</td>
<td>HTTP</td>
<td>210</td>
</tr>
<tr>
<td>NEXT_HTTP_VARIABLE function</td>
<td>HTTP</td>
<td>211</td>
</tr>
<tr>
<td>NOW function</td>
<td>Date and time</td>
<td>211</td>
</tr>
<tr>
<td>NTILE function</td>
<td>Analytical</td>
<td>212</td>
</tr>
<tr>
<td>NULLIF function</td>
<td>Miscellaneous</td>
<td>213</td>
</tr>
<tr>
<td>NUMBER function</td>
<td>Miscellaneous</td>
<td>214</td>
</tr>
<tr>
<td>OBJECT_ID function</td>
<td>System</td>
<td>215</td>
</tr>
<tr>
<td>OBJECT_NAME function</td>
<td>System</td>
<td>216</td>
</tr>
<tr>
<td>OCTET_LENGTH function</td>
<td>String</td>
<td>216</td>
</tr>
<tr>
<td>PATINDEX function</td>
<td>String</td>
<td>217</td>
</tr>
<tr>
<td>PERCENT_RANK function</td>
<td>Analytical</td>
<td>218</td>
</tr>
<tr>
<td>PERCENTILE_CONT function</td>
<td>Analytical</td>
<td>220</td>
</tr>
<tr>
<td>PERCENTILE_DISC function</td>
<td>Analytical</td>
<td>222</td>
</tr>
<tr>
<td>PI function</td>
<td>Numeric</td>
<td>224</td>
</tr>
<tr>
<td>POWER function</td>
<td>Numeric</td>
<td>224</td>
</tr>
<tr>
<td>PROPERTY function</td>
<td>System</td>
<td>225</td>
</tr>
<tr>
<td>PROPERTY_DESCRIPTION function</td>
<td>System</td>
<td>225</td>
</tr>
<tr>
<td>PROPERTY_NAME function</td>
<td>System</td>
<td>226</td>
</tr>
<tr>
<td>PROPERTY_NUMBER function</td>
<td>System</td>
<td>227</td>
</tr>
<tr>
<td>QUARTER function</td>
<td>Date and time</td>
<td>227</td>
</tr>
<tr>
<td>RADIANS function</td>
<td>Numeric</td>
<td>228</td>
</tr>
<tr>
<td>RAND function</td>
<td>Numeric</td>
<td>228</td>
</tr>
<tr>
<td>RANK function</td>
<td>Analytical</td>
<td>229</td>
</tr>
<tr>
<td>REGR_AVGX function</td>
<td>Aggregate</td>
<td>230</td>
</tr>
<tr>
<td>REGR_AVGY function</td>
<td>Aggregate</td>
<td>232</td>
</tr>
<tr>
<td>REGR_COUNT function</td>
<td>Aggregate</td>
<td>233</td>
</tr>
<tr>
<td>REGR_INTERCEPT function</td>
<td>Aggregate</td>
<td>234</td>
</tr>
<tr>
<td>REGR_R2 function</td>
<td>Aggregate</td>
<td>235</td>
</tr>
<tr>
<td>REGR_SLOPE function</td>
<td>Aggregate</td>
<td>236</td>
</tr>
<tr>
<td>REGR_SXX function</td>
<td>Aggregate</td>
<td>237</td>
</tr>
<tr>
<td>REGR_SXY function</td>
<td>Aggregate</td>
<td>238</td>
</tr>
<tr>
<td>REGR_SYY function</td>
<td>Aggregate</td>
<td>240</td>
</tr>
<tr>
<td>REMAINDER function</td>
<td>Numeric</td>
<td>241</td>
</tr>
<tr>
<td>REPEAT function</td>
<td>String</td>
<td>241</td>
</tr>
<tr>
<td>REPLACE function</td>
<td>String</td>
<td>242</td>
</tr>
<tr>
<td>REPLICATE function</td>
<td>String</td>
<td>243</td>
</tr>
<tr>
<td>REVERSE function</td>
<td>String</td>
<td>244</td>
</tr>
<tr>
<td>RIGHT function</td>
<td>String</td>
<td>245</td>
</tr>
<tr>
<td>ROUND function</td>
<td>Numeric</td>
<td>245</td>
</tr>
<tr>
<td>ROWID function</td>
<td>Miscellaneous</td>
<td>246</td>
</tr>
<tr>
<td>RTRIM function</td>
<td>String</td>
<td>247</td>
</tr>
</tbody>
</table>
SECOND function [Date and time] ........................................ 248
SECONDS function [Date and time] ...................................... 248
SIGN function [Numeric] .................................................... 249
SIMILAR function [String] .................................................. 250
SIN function [Numeric] ...................................................... 250
SORTKEY function [String] ................................................ 251
SOUNDEX function [String] ............................................... 255
SPACE function [String] ................................................... 256
SQRT function [Numeric] ................................................... 256
SQUARE function [Numeric] ............................................... 257
STDDEV function [Aggregate] ............................................. 257
STDDEV_POP function [Aggregate] ...................................... 258
STDDEV_SAMP function [Aggregate] ................................... 259
STR function [String] ....................................................... 260
STRING function [String] .................................................. 263
STRTOUUID function [String] ............................................ 263
STUFF function [String] ................................................... 264
SUBSTRINGS function [String] .......................................... 265
SUM function [Aggregate] .................................................. 265
SUSER_ID function [System] .............................................. 266
SUSER_NAME function [System] ......................................... 266
TAN function [Numeric] .................................................... 267
TODAY function [Date and time] ......................................... 267
TRIM function [String] ...................................................... 268
TRUNCNUM function [Numeric] ......................................... 268
TS_ARMA_AR function [Time Series] .................................... 269
TS_ARMA_CONST function [Time Series] ............................. 272
TS_ARMA_MA function [Time Series] .................................. 276
TS_AUTOCORRELATION function [Time Series] .................... 280
TS_AUTO_UNI_AR function [Time Series] ............................. 283
TS_BOX_COX_XFORM function [Time Series] ...................... 287
TS_DIFFERENCE function [Time Series] .............................. 289
TS_ESTIMATE_MISSING function [Time Series] .................... 294
TS_LACK OF FIT function [Time Series] ............................. 297
TS_LACK OF FIT_P function [Time Series] .......................... 301
TS_MAX_ARMA_AR function [Time Series] ............................ 304
TS_MAX_ARMA_CONST function [Time Series] ...................... 308
TS_MAX_ARMA_LIKELIHOOD function [Time Series] .............. 311
TS_MAX_ARMA_MA function [Time Series] ............................ 315
TS_OUTLIER_IDENTIFICATION function [Time Series] ........... 318
TS_PARTIAL_AUTOCORRELATION function [Time Series] ...... 323
TS_VWAP function [Time Series] ....................................... 327
UCASE function [String] .................................................. 329
Contents

sp_iqdatatype procedure ....................................................... 384
sp_iqdbsize procedure .......................................................... 386
sp_iqdbspace procedure ....................................................... 388
sp_iqdbspaceinfo procedure ................................................. 391
sp_iqdbspaceobjectinfo procedure ........................................ 394
sp_iqdbstatistics procedure ................................................... 397
sp_iqdropllogin procedure .................................................... 399
sp_iqemptyfile procedure ...................................................... 399
sp_iqestjoin procedure .......................................................... 400
sp_iqestdbspaces procedure ................................................ 402
sp_iqestspace procedure ...................................................... 404
sp_iqevent procedure ............................................................ 404
sp_iqfile procedure ................................................................ 407
sp_iqhelp procedure .............................................................. 408
sp_iqindex and sp_iqindex_alt procedures ........................... 415
sp_iqindexadvisor procedure .................................................. 418
sp_iqindexfragmentation procedure ....................................... 419
sp_iqindexinfo procedure ...................................................... 421
sp_iqindexmetadata procedure ............................................. 423
sp_iqindexsize procedure...................................................... 425
sp_iqindexuse procedure ...................................................... 426
sp_iqjoinindex procedure ...................................................... 427
sp_iqjoinindexsize procedure ................................................ 431
sp_iqlmconfig procedure ....................................................... 432
sp_iqlocks procedure ............................................................ 434
sp_iqmodifyadmin procedure ................................................ 437
sp_iqmodifylogin procedure ................................................... 438
sp_iqobjectinfo procedure ..................................................... 438
sp_iqpassword procedure ..................................................... 441
sp_iqkeys procedure ............................................................. 442
sp_iqprocedure procedure .................................................... 443
sp_iqproccparm procedure .................................................... 446
sp_iqrebuildindex procedure ................................................. 450
sp_iqrename procedure ........................................................ 452
sp_iq_reset_identity procedure ............................................. 454
sp_iqrestoreaction procedure ................................................ 454
sp_iqrowdensity procedure ................................................... 456
sp_iqshowpsexe procedure ................................................... 457
sp_iqspaceinfo procedure ..................................................... 459
sp_iqspaceused procedure .................................................... 460
sp_iqstatistics procedure ..................................................... 461
sp_iqstatus procedure ........................................................... 463
sp_iqsysmon procedure ........................................................ 466
sp_iqtable procedure ............................................................. 472
<table>
<thead>
<tr>
<th>Reference: Building Blocks, Tables, and Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSARTICLE system view .......................................................... 532</td>
</tr>
<tr>
<td>SYSARTICLECOL system view ......................................................... 532</td>
</tr>
<tr>
<td>SYSARTICLECOLS consolidated view ........................................ 532</td>
</tr>
<tr>
<td>SYSARTICLES consolidated view ................................................ 532</td>
</tr>
<tr>
<td>SYSCAPABILITIES consolidated view ........................................ 533</td>
</tr>
<tr>
<td>SYSCAPABILITY system view ..................................................... 533</td>
</tr>
<tr>
<td>SYSCAPABILITYNAME system view .............................................. 533</td>
</tr>
<tr>
<td>SYSCATALOG consolidated view .............................................. 534</td>
</tr>
<tr>
<td>SYSCHECK system view .............................................................. 534</td>
</tr>
<tr>
<td>SYSCOLAUTH consolidated view .................................................. 534</td>
</tr>
<tr>
<td>SYSCOLPERM system view ............................................................ 535</td>
</tr>
<tr>
<td>SYSCOLLATION compatibility view (deprecated) ........................ ... 535</td>
</tr>
<tr>
<td>SYSCOLLATIONMAPPINGS compatibility view (deprecated) ... .... 536</td>
</tr>
<tr>
<td>SYSCOLSTAT system view .......................................................... 536</td>
</tr>
<tr>
<td>SYSCOLSTATS consolidated view .............................................. 536</td>
</tr>
<tr>
<td>SYSCOLUMNS compatibility view (deprecated) ........................... 537</td>
</tr>
<tr>
<td>SYSCOLUMNS consolidated view .............................................. 537</td>
</tr>
<tr>
<td>SYSCOLUMNS ASE compatibility view ........................................... 537</td>
</tr>
<tr>
<td>SYSCOMMENTS ASE compatibility view .......................................... 538</td>
</tr>
<tr>
<td>SYSCONSTRAINT system view .................................................... 538</td>
</tr>
<tr>
<td>SYSDBFIELD system view .............................................................. 538</td>
</tr>
<tr>
<td>SYSDBSPACE system view ............................................................ 539</td>
</tr>
<tr>
<td>SYSDBSPACEPERM system view .................................................... 539</td>
</tr>
<tr>
<td>SYSDependency system view ......................................................... 539</td>
</tr>
<tr>
<td>SYSDomain system view ............................................................... 540</td>
</tr>
<tr>
<td>SYSEVENT system view ................................................................. 540</td>
</tr>
<tr>
<td>SYSEVENTTYPE system view ........................................................... 540</td>
</tr>
<tr>
<td>SYSEXTERNENV system view ........................................................... 541</td>
</tr>
<tr>
<td>SYSEXTERNENVOBJECT system view ............................................... 541</td>
</tr>
<tr>
<td>SYSEXTERNLOGIN system view ........................................................ 541</td>
</tr>
<tr>
<td>SYSFILE compatibility view (deprecated) ................................................ 542</td>
</tr>
<tr>
<td>SYSFKCOL compatibility view (deprecated) ........................................ 542</td>
</tr>
<tr>
<td>SYSFKKEY system view ................................................................. 542</td>
</tr>
<tr>
<td>SYSFOREIGNKEY compatibility view (deprecated) ................................. 543</td>
</tr>
<tr>
<td>SYSFOREIGNKEYS consolidated view ................................................ 543</td>
</tr>
<tr>
<td>SYSGROUP system view ............................................................... 543</td>
</tr>
<tr>
<td>SYSGROUPS consolidated view ...................................................... 544</td>
</tr>
<tr>
<td>SYSINDEX system view ................................................................. 544</td>
</tr>
<tr>
<td>SYSIDX system view .................................................................... 544</td>
</tr>
<tr>
<td>SYSIDXCOL system view ................................................................. 545</td>
</tr>
<tr>
<td>SYSINDEX compatibility view (deprecated) ........................................ 545</td>
</tr>
<tr>
<td>SYSINDEXES consolidated view ...................................................... 546</td>
</tr>
<tr>
<td>SYSINDEXES ASE compatibility view ................................................ 546</td>
</tr>
<tr>
<td>SYSINFO compatibility view (deprecated) ........................................ 546</td>
</tr>
<tr>
<td>System View Name</td>
</tr>
<tr>
<td>------------------------------------------</td>
</tr>
<tr>
<td>SYSIQBACKUPHISTORY</td>
</tr>
<tr>
<td>SYSIQBACKUPHISTORYDETAIL</td>
</tr>
<tr>
<td>SYSIQCOLUMN</td>
</tr>
<tr>
<td>SYSIQDBFILE</td>
</tr>
<tr>
<td>SYSIQDBSPACE</td>
</tr>
<tr>
<td>SYSIQFILE</td>
</tr>
<tr>
<td>SYSIQINFO</td>
</tr>
<tr>
<td>SYSIQJOINIDX</td>
</tr>
<tr>
<td>SYSIQJOININDEX</td>
</tr>
<tr>
<td>SYSIQJOINIXCOLUMN</td>
</tr>
<tr>
<td>SYSIQJOINIXTABLE</td>
</tr>
<tr>
<td>SYSIQMPXLOGPOLICYOPTION</td>
</tr>
<tr>
<td>SYSIQMPXSERVER</td>
</tr>
<tr>
<td>SYSIQOBJECTS ASE compatibility view</td>
</tr>
<tr>
<td>SYSIQPARTITIONCOLUMN</td>
</tr>
<tr>
<td>SYSIQTAB</td>
</tr>
<tr>
<td>SYSIQTABCOL</td>
</tr>
<tr>
<td>SYSIQTABLE</td>
</tr>
<tr>
<td>SYSIQVINDEX ASE compatibility view</td>
</tr>
<tr>
<td>SYSIXCOL compatibility view</td>
</tr>
<tr>
<td>SYSJAR</td>
</tr>
<tr>
<td>SYSJARCOMPONENT</td>
</tr>
<tr>
<td>SYSJAVACLASS</td>
</tr>
<tr>
<td>SYSLOGINMAP</td>
</tr>
<tr>
<td>SYSLOGINPOLICY</td>
</tr>
<tr>
<td>SYSLOGINPOLICYOPTION</td>
</tr>
<tr>
<td>SYSLOGINS ASE compatibility view</td>
</tr>
<tr>
<td>SYSMVOPTION</td>
</tr>
<tr>
<td>SYSMVOPTIONNAME</td>
</tr>
<tr>
<td>SYSOBJECT</td>
</tr>
<tr>
<td>SYSOBJECTS ASE compatibility view</td>
</tr>
<tr>
<td>SYSOPTION</td>
</tr>
<tr>
<td>SYSOPTIONS consolidated view</td>
</tr>
<tr>
<td>SYSOPTSTAT</td>
</tr>
<tr>
<td>SYSPARTITION</td>
</tr>
<tr>
<td>SYSPARTITIONKEY</td>
</tr>
<tr>
<td>SYSPARTITIONSCHHEME</td>
</tr>
<tr>
<td>SYSPHYSIDX</td>
</tr>
<tr>
<td>SYSPROCAUTH consolidated view</td>
</tr>
<tr>
<td>SYSPROCEDURE</td>
</tr>
<tr>
<td>SYSPROCOPARM</td>
</tr>
<tr>
<td>SYSPROCOPARMS consolidated view</td>
</tr>
<tr>
<td>SYSPROCOPERM</td>
</tr>
<tr>
<td>Reference: Building Blocks, Tables, and Procedures</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>SYSPROCS consolidated view .......................... 570</td>
</tr>
<tr>
<td>SYSPROXYTAB system view .................................. 570</td>
</tr>
<tr>
<td>SYSPUBLICATION system view .................................. 570</td>
</tr>
<tr>
<td>SYSPUBLICATIONS consolidated view ....................... 571</td>
</tr>
<tr>
<td>SYSSCHEDULE system view ....................................... 574</td>
</tr>
<tr>
<td>SYSSERVER system view ......................................... 574</td>
</tr>
<tr>
<td>SYSSOURCE system view .......................................... 575</td>
</tr>
<tr>
<td>SYSTAB system view .................................................. 579</td>
</tr>
<tr>
<td>SYSTABLE compatibility view (deprecated) ............ 579</td>
</tr>
<tr>
<td>SYSTABLEAUTH consolidated view ......................... 579</td>
</tr>
<tr>
<td>SYSTABCOL system view ............................................ 580</td>
</tr>
<tr>
<td>SYSTABLEPERM system view ...................................... 580</td>
</tr>
<tr>
<td>SYSTEXTIDX system view .......................................... 580</td>
</tr>
<tr>
<td>SYSTEXTIDXTAB system view .................................... 581</td>
</tr>
<tr>
<td>SYSTRIGGER system view .......................................... 581</td>
</tr>
<tr>
<td>SYSTRIGGERTAB system view .................................... 581</td>
</tr>
<tr>
<td>SYSTYPES ASE compatibility view ......................... 582</td>
</tr>
<tr>
<td>SYSTYPES AUTH compatibility view (deprecated) ........ 583</td>
</tr>
<tr>
<td>SYSTYPES AUTHORITY system view ......................... 583</td>
</tr>
<tr>
<td>SYSTYPES AUTHLIST compatibility view (deprecated) .... 584</td>
</tr>
</tbody>
</table>
Contents

Setting options for Transact-SQL compatibility ............................................. 616
Data manipulation language ........................................................................ 616
General guidelines for writing portable SQL .............................................. 617
Writing compatible queries ......................................................................... 617
Subqueries ...................................................................................................... 618
GROUP BY clause ...................................................................................... 619
COMPUTE clause ....................................................................................... 619
WHERE clause ........................................................................................... 619
Joins ............................................................................................................ 620
Null comparisons ........................................................................................ 621
Zero-length strings .................................................................................... 621
HOLDLOCK, SHARED, and FOR BROWSE ............................................... 621
SQL functions .......................................................................................... 622
OLAP functions ....................................................................................... 623
System functions ....................................................................................... 624
User-defined functions .............................................................................. 625
Arithmetic expressions on dates ............................................................... 625
SELECT INTO .......................................................................................... 626
Updatable views ......................................................................................... 626
FROM clause in UPDATE and DELETE .................................................... 626
Transact-SQL procedure language overview ............................................. 626
Transact-SQL stored procedure overview .................................................. 627
Transact-SQL batch overview .................................................................... 627
SQL statements in procedures and batches .............................................. 628
Automatic translation of stored procedures .............................................. 630
Returning result sets from Transact-SQL procedures ................................ 630
Variables in Transact-SQL procedures ..................................................... 631
Error handling in Transact-SQL procedures ............................................. 632
Using the RAISERROR statement in procedures ..................................... 633
Transact-SQL-like error handling in the Watcom-SQL dialect .................. 634
SQL Anywhere and Sybase IQ ................................................................. 634
Server and database startup and administration ..................................... 635
Database options ...................................................................................... 635
Data definition language (DDL) ............................................................... 636
Data manipulation language (DML) ........................................................ 637
Stored procedures ..................................................................................... 637
Adaptive Server Enterprise and Sybase IQ .............................................. 638
Stored procedures ..................................................................................... 638
System views ........................................................................................... 639

Index ............................................................................................................. 641
About This Book

Subject
This book provides reference material for many aspects of Sybase IQ, including SQL statements, language elements, data types, functions, system procedures, and system tables. Other books provide more context on how to perform particular tasks. This reference book is the place to look for information such as available SQL syntax, parameters, and options. For command line utility startup parameters, see the Utility Guide.

Audience
This manual is a reference for all users of Sybase IQ.

Related documents
The Sybase IQ 15.1 documentation set includes:

- *Release Bulletin* provides information about last-minute changes to the product and documentation.

- *Installation and Configuration Guide* provides platform-specific instructions on installing, migrating to a new version, and configuring Sybase IQ for a particular platform.

- *Advanced Security in Sybase IQ* covers the use of user encrypted columns within the Sybase IQ data repository. You need a separate license to install this product option.

- *Error Messages* lists Sybase IQ error messages referenced by Sybase error code, SQLCode, and SQLState, and SQL preprocessor errors and warnings.


- *Introduction to Sybase IQ* includes hands-on exercises for those unfamiliar with Sybase IQ or with the Sybase Central™ database management tool.

- *Large Objects Management in Sybase IQ* explains storage and retrieval of Binary Large Objects (BLOBs) and Character Large Objects (CLOBs) within the Sybase IQ data repository. You need a separate license to install this product option.
• *New Features in Sybase IQ 15.0* documents new features and behavior changes for version 15.0.

• *New Features Summary Sybase IQ 15.1* summarizes new features and behavior changes for the current version.

• *Performance and Tuning Guide* describes query optimization, design, and tuning issues for very large databases.

• *Quick Start* lists steps to build and query the demo database provided with Sybase IQ for validating the Sybase IQ software installation. Includes information on converting the demo database to multiplex.

• *Reference Manual* – includes two reference guides to Sybase IQ:
  - *Reference: Building Blocks, Tables, and Procedures* (this book) describes SQL, stored procedures, data types, and system tables that Sybase IQ supports.
  - *Reference: Statements and Options* describes the SQL statements and options that Sybase IQ supports.

• *System Administration Guide* – includes two volumes:
  - *System Administration Guide: Volume 1* describes startup, connections, database creation, population and indexing, versioning, collations, system backup and recovery, troubleshooting, and database repair.
  - *System Administration Guide: Volume 2* describes how to write and run procedures and batches, programming with OLAP, accessing remote data, setting up IQ as an Open Server, scheduling and event handling, programming with XML, and debugging.

• *User-Defined Functions Guide* provides information about the user-defined functions, their parameters, and possible usage scenarios.

• *Using Sybase IQ Multiplex* tells how to use multiplex capability, designed to manage large query loads across multiple nodes.

• *Utility Guide* provides Sybase IQ utility program reference material, such as available syntax, parameters, and options.
Sybase IQ and SQL Anywhere
Because Sybase IQ is an extension of SQL Anywhere®, a component of the SQL Anywhere package, Sybase IQ supports many of the same features as SQL Anywhere. The Sybase IQ documentation set refers you to SQL Anywhere documentation where appropriate.

Documentation for SQL Anywhere includes:

- **SQL Anywhere Server – Database Administration** describes how to run, manage, and configure SQL Anywhere databases. It describes database connections, the database server, database files, backup procedures, security, high availability, and replication with Replication Server®, as well as administration utilities and options.

- **SQL Anywhere Server – Programming** describes how to build and deploy database applications using the C, C++, Java, PHP, Perl, Python, and .NET programming languages such as Visual Basic and Visual C#. This book also describes a variety of programming interfaces such as ADO.NET and ODBC.

- **SQL Anywhere Server – SQL Reference** provides reference information for system procedures, and the catalog (system tables and views). It also provides an explanation of the SQL Anywhere implementation of the SQL language (search conditions, syntax, data types, and functions).

- **SQL Anywhere Server – SQL Usage** describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.

You can also refer to the SQL Anywhere documentation in the SQL Anywhere 11.0.1 collection at Product Manuals at http://www.sybase.com/support/manuals/ and in DocCommentXchange at http://dcx.sybase.com/dcx_home.php.

Documentation for Sybase Software Asset Management (SySAM) includes:

- **Sybase Software Asset Management (SySAM) 2** introduces asset management concepts and provides instructions for establishing and administering SySAM 2 licenses.

- **SySAM 2 Quick Start Guide** tells you how to get your SySAM-enabled Sybase product up and running.

- **FLEXnet Licensing End User Guide** explains FLEXnet Licensing for administrators and end users and describes how to use the tools that are part of the standard FLEXnet Licensing distribution kit from Sybase.
Other sources of information

Use the Sybase Getting Started CD, the SyBooks™ CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.

- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

Refer to the SyBooks Installation Guide on the Getting Started CD, or the README.txt file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

To access the Sybase Product Manuals Web site, go to Product Manuals at http://sybooks.sybase.com.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications


2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

3. Select Search to display the availability and certification report for the selection.

❖ Finding the latest information on component certifications

2 Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

3 Select Search to display the availability and certification report for the selection.

❖ Creating a personalized view of the Sybase Web site (including support pages)
Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ Finding the latest information on EBFs and software maintenance


2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Syntax conventions
This documentation uses these conventions in syntax descriptions:

• **Keywords** SQL keywords are shown in UPPERCASE. However, SQL keywords are case-insensitive, so you can enter keywords in any case; SELECT, Select, and select are equivalent.
• **Placeholders**  Items that must be replaced with appropriate identifiers or expressions are shown in *italics*.

• **Continuation**  Lines beginning with an ellipsis (...) are a continuation of the statements from the previous line.

• **Repeating items**  Lists of repeating items are shown with an element of the list followed by an ellipsis (...). One or more list elements are allowed. If multiple elements are specified, they must be separated by commas.

• **Optional portions**  Optional portions of a statement are enclosed by square brackets. For example:

  ```plaintext
  RELEASE SAVEPOINT [ savepoint-name ]
  ```

  The square brackets indicate that the *savepoint-name* is optional. Do not type the brackets.

• **Options**  When none or only one of a list of items must be chosen, the items are separated by vertical bars and the list enclosed in square brackets. For example:

  ```plaintext
  [ ASC | DESC ]
  ```

  The square brackets indicate that you can choose ASC, DESC, or neither. Do not type the brackets.

• **Alternatives**  When precisely one of the options must be chosen, the alternatives are enclosed in curly braces. For example:

  ```plaintext
  QUOTES { ON | OFF }
  ```

  The curly braces indicate that you must include either ON or OFF. Do not type the brackets.

**Typographic conventions**  Table 1 lists the typographic conventions used in this documentation.
The demo database

Sybase IQ includes scripts to create a demo database (iqdemo.db). Many of the queries and code samples in this document use the demo database as a data source.

The demo database contains internal information about a small company (employees, departments, and financial data), as well as product (products), and sales information (sales orders, customers, and contacts).

See the Sybase IQ installation guide for your platform or talk to your system administrator for more information about the demo database.

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Sybase IQ 15.1 and the HTML documentation have been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

Configuring your accessibility tool

You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool for information on using screen readers.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

Table 1: Typographic conventions

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>SQL and program code appears in a monospaced (fixed-width) font.</td>
</tr>
<tr>
<td>User entry</td>
<td>Text entered by the user is shown in a monospaced (fixed-width) font.</td>
</tr>
<tr>
<td>emphasis</td>
<td>Emphasized words are shown in italic.</td>
</tr>
<tr>
<td>file names</td>
<td>File names are shown in italic.</td>
</tr>
<tr>
<td>database objects</td>
<td>Names of database objects, such as tables and procedures, are shown in bold, sans serif type in print, and in italic online.</td>
</tr>
</tbody>
</table>
For a Section 508 compliance statement for Sybase IQ, go to Sybase Accessibility at http://www.sybase.com/products/accessibility.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.
CHAPTER 1

File Locations and Installation Settings

About this chapter

This chapter describes the installation and operating system settings used by Sybase IQ. Depending on the operating system, these settings may be stored as environment variables, initialization file entries, or registry entries.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation directory structure</td>
<td>2</td>
</tr>
<tr>
<td>How Sybase IQ locates files</td>
<td>3</td>
</tr>
<tr>
<td>Environment variables</td>
<td>5</td>
</tr>
<tr>
<td>Registry entries</td>
<td>14</td>
</tr>
</tbody>
</table>
When you install Sybase IQ, several directories may be created. The directories created depend on which options you choose during installation and which directories already exist in your Sybase directory (the directory defined by $SYBASE on UNIX or %SYBASE% on Windows). This section describes the directory structure.

By default, Sybase IQ software is installed in a unique subdirectory under the Sybase directory. This subdirectory is called the installation directory. Other tools provided with Sybase IQ have unique subdirectories under the Sybase directory. This section describes only the subdirectory structure for Sybase IQ.

By default, the Sybase IQ directory is IQ-15_1. The location of IQ-15_1 varies, depending on where you install Sybase IQ. The IQ-15_1 directory is also referred to by the environment variable $IQDIR15 on UNIX or %IQDIR15% on Windows.

The Sybase IQ directory holds a number of directories and files:

- **Demo directory** (%ALLUSERSPROFILE%/SybaseIQ/demo) – holds the tools required to build the iqdemo database. The iqdemo database files are iqdemo.db, iqdemo.iq, iqdemo.iqmsg, and iqdemo.iqtmp. The demo database itself is not shipped with Sybase IQ.

  The subdirectory /demo/adata holds 15.x data to allow the creation of the 15.x iqdemo database. The subdirectory /demo/demodata holds Sybase IQ 12.7 data to allow the creation of an iqdemo database that has the same schema layout and data as the IQ 12.7 asiqdemo database. Use /demo/mkiqdemo.bat on Windows and demo/mkiqdemo.sh on Unix to create the 15.x iqdemo database. The iqdemo database can be used to demonstrate problems to Technical Support.

- **Scripts directory** (IQ-15_1/scripts) – holds some scripts used in examples and when creating catalog objects like stored procedures. *Do not edit these scripts.* If you edit, delete, or move these scripts, the server will not operate properly.
• **Samples directories** The `samples` directory contains SQL samples and user-defined function (UDF) samples. `%ALLUSERSPROFILE%/SybaseIQ/samples/sqlanywhere` contains 24 directories of SQL samples. The `sqlanywhere/c` directory holds C++ examples that illustrate using ESQL (embedded SQL) and C with SQL Anywhere. Because SQL Anywhere and Sybase IQ share common code, you can modify these examples for use with Sybase IQ. The `%ALLUSERSPROFILE%/SybaseIQ/samples/udf` directory holds sample C++ scalar and aggregate UDFs.

• **Executable directories** – hold executables, libraries, help files, and so on.

  On UNIX, executable subdirectories include `IQ-15_1` subdirectories /bin64, /lib64, /logfiles, /res, and /tix. On Windows, these include `IQ-15_1` subdirectories \h, \install, \java, and \bin32.

### How Sybase IQ locates files

To start and run, Sybase IQ must find and access several types of files. Understanding how Sybase IQ finds these files is important, to ensure that the correct files are used. Several directories or files with identical names may reside on a system. Sybase IQ uses both Adaptive Server™ Enterprise and SQL Anywhere libraries. If either of these products have already been installed on your system, note the directory where they are installed to avoid confusion.

The types of files include but are not limited to:

• **Libraries** – might include product libraries or system libraries. File name extensions include `.so.nnn` or `.so` on UNIX, or `.dll` or `.lib` on Windows. These files are required to run Sybase IQ. If an incorrect DLL is found, a version mismatch error may occur. For example, library files might be found in `SIQDIR15/lib64` or `$SYBASE/$SYBASE_OCS/lib64` on UNIX, or `%IQDIR15%/bin32` or `%SYBASE%SYBASE_OCS\dll` on Windows. An empty directory, `SIQDIR15/usrlib`, lets you supersede default libraries with custom libraries and patches, because `start_iq` includes `usrlib` before regular library directories.

• **Interface files** – required to run Sybase IQ. For example, `.odbc.ini` and `utility_db.ini` on UNIX, and `util_db.ini` on Windows. For more information about these files, see the *System Administration Guide: Volume 1* and the *Installation and Configuration Guide.*
How Sybase IQ locates files

- Configuration files – used to specify connection parameters. Examples include `default.cfg` on Windows or `iqdemo.cfg`.
- Database files – store the data and metadata. For example: `iqdemo.db`, `iqdemo.iq`, `iqdemo.iqmsg`, `iqdemo.iqtmp`.
- Log files – store information about the current session on the server and connected database. For example, a server log might be named `%ALLUSERSPROFILE%/SybaseIQ/IQ15_1/logfiles/yourservename.0006.srvlog`. The database log (for example, `%ALLUSERSPROFILE%/SybaseIQ/IQ-15_1/demo/iqdemo.log`) is created when you connect to the database. For more information about these files, see the Installation and Configuration Guide.
- Product scripts – are sample files that show how to create, populate, and upgrade databases.
- User files – include flat files used with the LOAD command and SQL scripts used with tools such as Interactive SQL.
- Temporary files – created by Sybase IQ to store temporary information for operations like performing sorts for queries.

Some file names are specified in SQL statements and must be located at runtime. Examples of SQL statements that use file names include the following:

- INSTALL statement – the name of the file that holds Java classes.
- LOAD TABLE statement – the name of the file from which data should be loaded.
- CREATE DATABASE statement – A file name is needed for this statement and similar statements that can create files.

In some cases, Sybase IQ uses a simple algorithm to locate files. In other cases, a more extensive search is carried out.

**Simple file searching**

In many SQL statements such as LOAD TABLE or CREATE DATABASE, the file name is interpreted as relative to the current working directory of the database server; that is, where the server was started.

Also, when a database server is started and a database file name (DBF parameter) is supplied, the path is interpreted as relative to the directory in which the server was started.
Extensive file searching

Sybase IQ programs, including the database server and administration utilities, carry out extensive searches for required files, such as DLLs or shared libraries. In these cases, Sybase IQ programs look for files in the following order:

1. The executable directory – the directory in which the program executable is held. Also, directories with the following paths relative to the program executable directory:
   - Parent of the executable directory.
   - A child of the parent directory named *scripts*.

2. Current working directory – when a program is started, it has a current working directory (the directory from which it is started). This directory is searched for required files.

3. Location registry entry – on a Windows installation, Sybase IQ adds a LOCATION registry entry. The indicated directory is searched, followed by the following:
   - A child named *scripts*
   - A child with the operating system name (*bin32*, *bin*, and so on)

4. System-specific directories – this includes directories where common operating system files are held, such as the Windows directory and the Windows\system directory on Windows.

5. CLASSPATH directories – for Java files, directories listed in the CLASSPATH environment variable are searched to locate files.

6. PATH directories – directories in the system path and the user’s path are searched to locate files.

7. LIBRARY PATH directories – directories listed in the *LIBPATH* environment variable are searched for shared libraries.

Environment variables

Sybase IQ uses environment variables to store various types of information; not all variables need to be set in all circumstances.
Setting environment variables

Required environment variables are set by environment source files on UNIX and by the Sybase IQ installation on Windows.

❖ Running UNIX environment source files

Issue the following command to set all required environment variables.

1 For the Bourne/Korn shell:
   . $SYBASE/IQ-15_1/IQ-15_1.sh
2 For the C shell:
   source $SYBASE/IQ-15_1/IQ-15_1.csh;
   rehash

On Windows platforms, the installation program automatically sets all environmental variables, so no changes are necessary. However, if you must set optional variables or change defaults, use one of the following procedures, as appropriate for your operating system.

❖ Setting environment variables on Windows

1 On your desktop, right-click My Computer and select Properties from the submenu.
2 Click the Advanced tab.
3 Click the Environment Variables button.

The Environment Variables dialog opens.

   a If the environment variable does not already exist, click New and type the variable name and its value in the spaces provided; then click OK.
   b If the variable does exist, select it from the list of System Variables or User Variables, click Edit, and make any modifications in the Variable Value field. Then click OK to capture the setting.

Note See the Microsoft Windows documentation for an explanation of user variables and system variables.

❖ Setting environment variables on UNIX

1 To check the setting for an environment variable, use:

   echo $variable-name

   For example, to see the setting for the $SYBASE variable:
In one of your startup files (.cshrc, .shrc, .login), add a line that sets the variable.

In some shells (such as sh, bash, ksh) the line is:

```
VARIABLE=value;export VARIABLE
```

In other shells (such as csh, tsch) the line is:

```
setenv VARIABLE "value"
```

For details about variables Sybase IQ uses, see:

- “IQDIR15 environment variable” on page 7
- “IQPORT environment variable” on page 8
- “IQLOGDIR15 environment variable” on page 8
- “IQTIMOUT environment variable” on page 9
- “IQTMP15 environment variable” on page 9
- “LIBPATH or LD_LIBRARY_PATH environment variable” on page 10
- “PATH environment variable” on page 11
- “SACHARSET environment variable” on page 11
- “SALANG environment variable” on page 11
- “SQLCONNECT environment variable” on page 12
- “SYBASE environment variable” on page 12
- “$SYBASE_JRE6_32, $SYBASE_JRE6_64, $SYBASE_JRE5_64 environment variables” on page 13
- “SYBASE_OCS environment variable” on page 13

### IQDIR15 environment variable

**Setting**

```
IQDIR15 = ${SYBASE}/IQ-15_1
```

**Operating system**

(Required) Set by the environment source file or the installation program. This default setting can be changed on Windows.

**Description**

IQDIR15 identifies the location of the Sybase IQ directory and is the location for other directories and files under that directory:
Environment variables

- $IQDIR15/bin[64]/util_db.ini holds the login ID and password for the utility database, utility_db. The installation program lets you change these from their default values, login ID “DBA” and password “sql.”

- $IQDIR15/logfiles is the default location for the server log and backup/restore log (the backup history file). You can override this default by setting the IQLOGDIR15 environment variable.

- $IQDIR15/demo is the location for the iqdemo database files.

**IQPORT environment variable**

**Setting**

IQPORT = 5556

**Operating system**

Optional. If the user did not specify IQPORT in the environment source file, the port number defaults to 1099. You can change this default value, provided you do so before the plug-in starts. You can set this variable as described in “Setting environment variables” on page 6 or by supplying the -DIQPORT argument to the scjview command when starting Sybase Central. For example:

`scjview -DIQPORT=3345`

**Description**

Overrides the default value for the Sybase IQ Agent port number, which is used for communications between the Sybase IQ plug-in and Agent.

**Note** Once the agent starts, you cannot change the port value.

1099 is the plug-in default value when searching for an agent process on any given port. If the plug-in finds no agent on this port, it displays a prompt so that you can specify the correct port value.

**IQLOGDIR15 environment variable**

**Setting**

IQLOGDIR15 = path

**Operating system**

Optional.

The IQLOGDIR15 environment variable is not set by the installation program. It defines the location of various log files:

- The server log is in the file `servername.nnen.srvlog` (where nnn is the number of times the server has been started) in the directory specified by $IQLOGDIR15.
If IQLOGDIR15 is not set to a valid, write-enabled directory, then most utilities, including start_iq, use the default location $IQDIR15/logfiles for all server logs.

**IQTLOAD environment variable**

<table>
<thead>
<tr>
<th>Setting</th>
<th>IQLOAD = nnn</th>
</tr>
</thead>
</table>
| Operating system | Optional but recommended in multiplex environments. For information on multiplex capability, see *Using Sybase IQ Multiplex*.
| Description | The Sybase IQ Agent waits indefinitely for a process to complete. Setting a wait time is recommended when creating or synchronizing query servers for a multiplex with a very large catalog store. Large catalog stores extend the time needed for the dbbackup part of synchronization, and increasing the wait time accommodates a larger synchronize.

IQTLOAD overrides the default wait time of five minutes, and the argument nnn is the number of minutes for the Sybase IQ Agent to wait. For example:

- To wait 45 minutes (Korn or Bourne shell):

  ```
  IQLOAD=45
  export IQLOAD
  ```

- To wait an hour (C shell):

  ```
  setenv IQLOAD 60
  ```

**Note** Set IQLOAD before you invoke the agent startup option. See “Before you Install” and “Starting the Sybase IQ Agent” in the *Installation and Configuration Guide* and “Running the Sybase IQ Agent” in *Introduction to Sybase IQ*.

**IQTMP15 environment variable**

<table>
<thead>
<tr>
<th>Setting</th>
<th>IQTMP15 = temp_directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Optional on UNIX. Not used on Windows platforms.</td>
</tr>
<tr>
<td>Description</td>
<td>The IQTMP15 environment variable is not set by the installation program. IQTMP15 is used by Sybase IQ to indicate a directory where temporary files are kept.</td>
</tr>
</tbody>
</table>
Environment variables

The IQTMP15 environment variable should point to a local directory for those using NFS (network file system), which permits the IQTMP15 directory to purge directories and files that are no longer needed as client connections are closed. Each client connection creates several directories and files in a temporary directory. These are needed only for the duration of the connection. The directory must have write permissions for all users who connect to the server.

Note The temporary files whose location is defined by IQTMP15 are files used by the client and server. This variable does not control the default location of your IQ temporary store. For information on how Sybase IQ determines the location of your temporary store, see the CREATE DATABASE statement in Chapter 1, “SQL Statements,” in Reference: Statements and Options.

If you do not explicitly set IQTMP15, or if it is set to $SYBASE or $IQDIR15, then the Sybase IQ Agent sets IQTMP15 to a subdirectory in the UNIX directory /tmp.

If more than one database server is running on a machine, each server and associated local client needs a separate temporary directory to avoid conflicts. When you do not specify a port or engine number for connection, Sybase IQ uses shared memory connectivity instead of network connectivity.

To avoid conflicts when using shared memory:

• Create a temporary directory dedicated to each server. Make sure that each local client uses the same temporary directory as its server by setting the IQTMP15 environment variable explicitly in both environments.

• Create a data source name in the .odbc.ini file (on UNIX) for each server and provide detailed connection information. See the Installation and Configuration Guide.

• Use connection strings that specify explicit parameters instead of relying on defaults.

• Confirm connections by issuing:

```sql
SELECT "database name is" = db_name(), "servername_is" = @@servername
```

LIBPATH or LD_LIBRARY_PATH environment variable

Settings For AIX:
**LIBPATH** = `installation_path/lib`

For all other UNIX/LINUX platforms:

LD_LIBRARY_PATH = `installation_path/lib`

**Description**

Required. Variable name is platform dependent. UNIX only.

Specifies the directories where Sybase IQ shared libraries are located. On UNIX, set the library path variable by running the environment source file.

---

**PATH environment variable**

**Setting**

`PATH = installation_path`

**Operating system**

Required.

**Description**

The PATH environment variable is an operating system required variable that includes the directories where Sybase IQ executables are located. On Windows, the installation program modifies PATH. On UNIX, run the environment source file to include the necessary directories.

On Windows, PATH takes the place of the LIBRARY_PATH variable, so executables and DLLs are located using the PATH variable.

---

**SACHARSET environment variable**

**Setting**

`SACHARSET=charset`

**Description**

Charset is a character set name. For example, setting SACHARSET=cp1252 sets the default character set to cp1252.

The first of the following values set determines the default character set:

- SACHARSET environment variable
- Query the operating system

If no character set information is specified, use iso_1 for UNIX, or cp850 otherwise.

---

**SALANG environment variable**

**Setting**

`SALANG=language_code`

**Operating system**

Optional but recommended in non-English environments.
Environment variables

**Language code** is the two-letter combination representing a language. For example, setting SALANG=DE sets the default language to German.

The first of the following values set determines the default language.

- SALANG environment variable
- Registry (Windows only) as set by the installer or dblang.exe
- Query the operating system

If no language information is set, English is the default.

**SQLCONNECT environment variable**

**Settings**

SQLCONNECT = parameter#value ; ...

**Operating system**

Optional.

**Description**

The SQLCONNECT environment variable is optional, and is not set by the installation program.

SQLCONNECT specifies connection parameters that are used by several of the database administration utilities, such as DBISQL, DBINFO, DBCOLLAT, and DBSTOP, when connecting to a database server. This string is a list of parameter settings, of the form parameter=value, delimited by semicolons.

The number sign “#” is an alternative to the equals sign; use it if you are setting the connection parameters string in the SQLCONNECT environment variable. Using “=” inside an environment variable setting is a syntax error. The = sign is allowed only in Windows.

**Note** Specifying connection parameters in SQLCONNECT rather than on the command line offers greater security on UNIX systems. It prevents users from being able to display your password with the ps -ef command. This is especially useful if you run DBISQL or other utilities in quiet mode.

**See also**

See “Connection parameters” in Chapter 4, “Connection and Communication Parameters,” in the *System Administration Guide: Volume 1*.

**SYBASE environment variable**

**Setting**

SYBASE = path
### Operating system
Required.

### Description
SYBASE identifies the location of Sybase applications, such as Open Client and Open Server. You must set the SYBASE variable before you can install Sybase IQ on UNIX systems. This variable is required for using Sybase Central on UNIX systems.

### $SYBASE_JRE6_32, $SYBASE_JRE6_64, $SYBASE_JRE5_64 environment variables

<table>
<thead>
<tr>
<th>Setting</th>
<th>SYBASE_JRE= &quot;${SYBASE}/shared/jre-6_0&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>
This variable specifies the location of the Java Runtime Environment used by the Sybase Central plug-in for Sybase IQ. For Windows and UNIX, the environment variable is $SYBASE_JRE6_32 or $SYBASE_JRE6_64. For AIX/LinuxIBM the variable is $SYBASE_JRE5_64.

On UNIX, run the `SYBASE.csh` (C shell) or `SYBASE.sh` (Bourne or Korn shell) environment source file. On Windows, the installation program sets the variable when it installs Open Client Software Developer’s Kit.

### SYBASE_OCS environment variable

<table>
<thead>
<tr>
<th>Setting</th>
<th>SYBASE_OCS = &quot;OCS-15_1&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Required.</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>
The SYBASE_OCS variable specifies the home directory for the Open Client product. This variable is only used on Window. On Windows, the installation program sets SYBASE_OCS when it installs Open Client/Server Software Developers Kit.
Registry entries

On Windows operating systems, Sybase IQ uses several Registry settings. These settings are made for you by the software, and in general operation, you should not need to access the registry. The settings are provided here if you modify your operating environment.

Warning! Sybase recommends not modifying the Registry, as incorrect changes might damage your system.

Current user and local machine settings

Some operating systems, such as Windows, hold two levels of system settings. Some settings are specific to an individual user and are used only when that user is logged on; these settings are called current user settings. Some settings are global to the machine and are available no matter which user is logged on; these are called local machine settings. You must have administrator permissions on your machine to make local machine settings.

Sybase IQ permits the use of both current user and local machine settings. For Windows, these settings are held in the HKEY_CURRENT_USER registry and HKEY_LOCAL_MACHINE registry, respectively.

The Sybase IQ installation lets you choose whether the settings it makes are for the current user only or at the local machine level.

If you make settings in both current user and local machine registries, the current user setting takes precedence over the local machine setting.

If you are running a Sybase IQ program as a service on Windows, ensure that the settings are made at the local machine level.

Services can continue to run under a special account when you log off a machine, as long as you do not shut the machine down entirely. Services can be made independent of individual accounts and need access to local machine settings.

In general, Sybase recommends using local machine settings.

Registry structure

On Windows, you can access the Registry directly using the Registry Editor.
To start the editor, select Start > Run and type in the Open box
gedit32

Note  Read Only Mode protects your Registry data from accidental changes. To use it, open the Registry Editor, select Edit >Permissions, and then check Read permission.

The Sybase IQ registry entry is held in the HKEY_LOCAL_MACHINE key, in the following location:

SOFTWARE
  Sybase
    IQ 15.0

Registry settings on installation

The installation program automatically makes the following registry settings in the Sybase registry:

- Location – In the Sybase IQ registry, this entry holds the installation directory location. For example:

  Location:REG_SZ:C:\Program Files\Sybase\IQ-15_1

The Sybase IQ registry includes other entries for installed applications. The Sybase Central registry holds information about the Sybase Central version and installed plug-ins.
CHAPTER 2

SQL Language Elements

About this chapter

This chapter presents detailed descriptions of the language elements and conventions of Sybase IQ SQL.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keywords</td>
<td>17</td>
</tr>
<tr>
<td>Identifiers</td>
<td>21</td>
</tr>
<tr>
<td>Strings</td>
<td>22</td>
</tr>
<tr>
<td>Expressions</td>
<td>23</td>
</tr>
<tr>
<td>Search conditions</td>
<td>33</td>
</tr>
<tr>
<td>Special values</td>
<td>53</td>
</tr>
<tr>
<td>Variables</td>
<td>56</td>
</tr>
<tr>
<td>Comments</td>
<td>65</td>
</tr>
<tr>
<td>NULL value</td>
<td>67</td>
</tr>
</tbody>
</table>

Keywords

Each SQL statement contains one or more keywords. SQL is not case-sensitive to keywords, but throughout these manuals, keywords are indicated in uppercase.

For example, in the following statement, SELECT and FROM are keywords:

```
SELECT *
FROM Employees
```

The following statements are equivalent to the one above:

```
Select *
From Employees
select * from Employees
sELECT * FRoM Employees
```
Keywords

Reserved words

Some keywords in SQL are also reserved words. To use a reserved word in a SQL statement as an identifier, you must enclose the word in double quotes. Many, but not all, of the keywords that appear in SQL statements are reserved words. For example, you must use the following syntax to retrieve the contents of a table named SELECT.

```
SELECT *
FROM "SELECT"
```

Because SQL is not case-sensitive with respect to keywords, each of the words in Table 2-1 may appear in uppercase, lowercase, or any combination of the two. All strings that differ only in capitalization from these words are reserved words.

If you are using Embedded SQL, you can use the database library function `sql_needs_quotes` to determine whether a string requires quotation marks. A string requires quotes if it is a reserved word or if it contains a character not ordinarily allowed in an identifier.

Table 2-1 lists the SQL reserved words in Sybase IQ.
### Table 2-1: SQL reserved words

<table>
<thead>
<tr>
<th>active</th>
<th>add</th>
<th>all</th>
<th>algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>alter</td>
<td>add</td>
<td>any</td>
<td>append</td>
</tr>
<tr>
<td>as</td>
<td>asc</td>
<td>auto</td>
<td>backup</td>
</tr>
<tr>
<td>begin</td>
<td>between</td>
<td>bigint</td>
<td>binary</td>
</tr>
<tr>
<td>bit</td>
<td>bottom</td>
<td>break</td>
<td>by</td>
</tr>
<tr>
<td>calibrate</td>
<td>calibration</td>
<td>call</td>
<td>cancel</td>
</tr>
<tr>
<td>capability</td>
<td>cascade</td>
<td>case</td>
<td>cast</td>
</tr>
<tr>
<td>certificate</td>
<td>char</td>
<td>char_convert</td>
<td>character</td>
</tr>
<tr>
<td>check</td>
<td>checkpoint</td>
<td>checksum</td>
<td>clientport</td>
</tr>
<tr>
<td>close</td>
<td>columns</td>
<td>comment</td>
<td>commit</td>
</tr>
<tr>
<td>committed</td>
<td>comparisons</td>
<td>computes</td>
<td>conflict</td>
</tr>
<tr>
<td>connect</td>
<td>constraint</td>
<td>contains</td>
<td>continue</td>
</tr>
<tr>
<td>convert</td>
<td>create</td>
<td>cross</td>
<td>cube</td>
</tr>
<tr>
<td>current</td>
<td>current_timestamp</td>
<td>current_user</td>
<td>cursor</td>
</tr>
<tr>
<td>date</td>
<td>dbspace</td>
<td>dbspacename</td>
<td>deallocate</td>
</tr>
<tr>
<td>debug</td>
<td>dec</td>
<td>decimal</td>
<td>declare</td>
</tr>
<tr>
<td>decoupled</td>
<td>decrypted</td>
<td>default</td>
<td>delay</td>
</tr>
<tr>
<td>delete</td>
<td>deleting</td>
<td>density</td>
<td>desc</td>
</tr>
<tr>
<td>deterministic</td>
<td>disable</td>
<td>distinct</td>
<td>do</td>
</tr>
<tr>
<td>double</td>
<td>drop</td>
<td>dynamic</td>
<td>elements</td>
</tr>
<tr>
<td>else</td>
<td>elseif</td>
<td>enable</td>
<td>encapsulated</td>
</tr>
<tr>
<td>encrypted</td>
<td>end</td>
<td>endif</td>
<td>escape</td>
</tr>
<tr>
<td>except</td>
<td>exception</td>
<td>exclude</td>
<td>exec</td>
</tr>
<tr>
<td>execute</td>
<td>existing</td>
<td>exists</td>
<td>explicit</td>
</tr>
<tr>
<td>express</td>
<td>externlogin</td>
<td>fastfirstrow</td>
<td>fetch</td>
</tr>
<tr>
<td>first</td>
<td>float</td>
<td>following</td>
<td>for</td>
</tr>
<tr>
<td>force</td>
<td>foreign</td>
<td>forward</td>
<td>from</td>
</tr>
<tr>
<td>full</td>
<td>gb</td>
<td>goto</td>
<td>grant</td>
</tr>
<tr>
<td>group</td>
<td>grouping</td>
<td>having</td>
<td>hidden</td>
</tr>
<tr>
<td>history</td>
<td>holdlock</td>
<td>identified</td>
<td>if</td>
</tr>
<tr>
<td>in</td>
<td>inactive</td>
<td>index</td>
<td>index_lparen</td>
</tr>
<tr>
<td>inner</td>
<td>inout</td>
<td>input</td>
<td>insensitive</td>
</tr>
<tr>
<td>insert</td>
<td>inserting</td>
<td>install</td>
<td>instead</td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>integrated</td>
<td>intersect</td>
</tr>
<tr>
<td>into</td>
<td>iq</td>
<td>is</td>
<td>isolation</td>
</tr>
<tr>
<td>jdk</td>
<td>join</td>
<td>kb</td>
<td>key</td>
</tr>
<tr>
<td>lateral</td>
<td>left</td>
<td>like</td>
<td>lock</td>
</tr>
</tbody>
</table>
### Keywords

<table>
<thead>
<tr>
<th>keyword</th>
<th>keyword</th>
<th>keyword</th>
<th>keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging</td>
<td>login</td>
<td>long</td>
<td>mb</td>
</tr>
<tr>
<td>match</td>
<td>membership</td>
<td>message</td>
<td>mode</td>
</tr>
<tr>
<td>modify</td>
<td>namespace</td>
<td>natural</td>
<td>new</td>
</tr>
<tr>
<td>no</td>
<td>noholdlock</td>
<td>nolock</td>
<td>not</td>
</tr>
<tr>
<td>notify</td>
<td>null</td>
<td>numeric</td>
<td>of</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
<td>open</td>
<td>optimization</td>
</tr>
<tr>
<td>option</td>
<td>options</td>
<td>or</td>
<td>order</td>
</tr>
<tr>
<td>others</td>
<td>out</td>
<td>outer</td>
<td>over</td>
</tr>
<tr>
<td>pages</td>
<td>paglock</td>
<td>partial</td>
<td>partition</td>
</tr>
<tr>
<td>passsthrough</td>
<td>password</td>
<td>plan</td>
<td>preceding</td>
</tr>
<tr>
<td>precision</td>
<td>prepare</td>
<td>primary</td>
<td>print</td>
</tr>
<tr>
<td>privileges</td>
<td>proc</td>
<td>procedure</td>
<td>proxy</td>
</tr>
<tr>
<td>publication</td>
<td>raiserror</td>
<td>range</td>
<td>raw</td>
</tr>
<tr>
<td>readcommitted</td>
<td>readonly</td>
<td>readpast</td>
<td>readtext</td>
</tr>
<tr>
<td>readuncommitted</td>
<td>readwrite</td>
<td>real</td>
<td>recursive</td>
</tr>
<tr>
<td>reference</td>
<td>references</td>
<td>release</td>
<td>relocate</td>
</tr>
<tr>
<td>remote</td>
<td>remove</td>
<td>rename</td>
<td>reorganize</td>
</tr>
<tr>
<td>repeatable</td>
<td>repeataberead</td>
<td>reserve</td>
<td>resizing</td>
</tr>
<tr>
<td>resource</td>
<td>restore</td>
<td>restrict</td>
<td>return</td>
</tr>
<tr>
<td>revoke</td>
<td>right</td>
<td>rollback</td>
<td>rollup</td>
</tr>
<tr>
<td>root</td>
<td>row</td>
<td>rowlock</td>
<td>rows</td>
</tr>
<tr>
<td>save</td>
<td>savepoint</td>
<td>schedule</td>
<td>scroll</td>
</tr>
<tr>
<td>secure</td>
<td>select</td>
<td>sensitive</td>
<td>serializable</td>
</tr>
<tr>
<td>service</td>
<td>session</td>
<td>set</td>
<td>setuser</td>
</tr>
<tr>
<td>share</td>
<td>smallint</td>
<td>soapaction</td>
<td>some</td>
</tr>
<tr>
<td>space</td>
<td>sqlcode</td>
<td>sqlstate</td>
<td>start</td>
</tr>
<tr>
<td>stop</td>
<td>subtrans</td>
<td>subtransaction</td>
<td>synchronize</td>
</tr>
<tr>
<td>syntax_error</td>
<td>table</td>
<td>tablock</td>
<td>tablockx</td>
</tr>
<tr>
<td>tb</td>
<td>temporary</td>
<td>then</td>
<td>ties</td>
</tr>
<tr>
<td>time</td>
<td>timestamp</td>
<td>tinyint</td>
<td>to</td>
</tr>
<tr>
<td>top</td>
<td>tran</td>
<td>transaction</td>
<td>transactional</td>
</tr>
<tr>
<td>transfer</td>
<td>tries</td>
<td>trigger</td>
<td>truncate</td>
</tr>
<tr>
<td>tsequal</td>
<td>unbounded</td>
<td>uncommitted</td>
<td>union</td>
</tr>
<tr>
<td>unique</td>
<td>uniqueidentifier</td>
<td>unknown</td>
<td>unsigned</td>
</tr>
<tr>
<td>update</td>
<td>updating</td>
<td>uplock</td>
<td>url</td>
</tr>
<tr>
<td>user</td>
<td>utc</td>
<td>using</td>
<td>validate</td>
</tr>
<tr>
<td>values</td>
<td>varbinary</td>
<td>varchar</td>
<td>variable</td>
</tr>
<tr>
<td>varying</td>
<td>virtual</td>
<td>view</td>
<td>wait</td>
</tr>
</tbody>
</table>
**Identifiers**

**Function**

Identifiers are names of objects in the database, such as user IDs, tables, and columns.

**Description**

Identifiers have a maximum length of 128 bytes. They must be enclosed in double quotes or square brackets if any of the following conditions are true:

- The identifier contains spaces.
- The first character of the identifier is not an alphabetic character (as defined below).
- The identifier contains a reserved word.
- The identifier contains characters other than alphabetic characters and digits.

*Alphabetic characters* include the alphabet, as well as the underscore character (_), at sign (@), number sign (#), and dollar sign ($). The database collation sequence dictates which characters are considered alphabetic or digit characters.

You can represent an apostrophe (single quote) inside an identifier by following it with another apostrophe.

Identifiers have the following limitations:

- Table names cannot contain double quotes.
- User names and database names cannot contain double quote, single quote, and semi-colon characters.
- User names and database names cannot start or end with a space.
- Dbspace names are not case-sensitive in a CASE RESPECT database.

---

waitfor | web | when | where |
--------|-----|------|-------|
while   | window | with | withauto |
with_cube | with_lparen | with_rollup | within |
word    | work   | writeserver | writetext |
xlock   | xml     |
Strings

If the QUOTED_IDENTIFIER database option is set to OFF, double quotes are used to delimit SQL strings and cannot be used for identifiers. However, you can always use square brackets to delimit identifiers, regardless of the setting of QUOTED_IDENTIFIER.

The default setting for the QUOTED_IDENTIFIER option is OFF for Open Client and jConnect connections; otherwise the default is ON.

Examples

The following are all valid identifiers.

Surname
"Surname"
[Surname]
SomeBigName
"Client Number"

See also

For a complete list of reserved words, see “Reserved words” on page 18.

For information on the QUOTED_IDENTIFIER option, see “The quoted_identifier option” on page 32.

For additional restrictions on server and database names, see “Server command-line switches” on page 7 in Chapter 1, “Running the Database Server” of the Utility Guide.

Strings

Strings are of the following types:

- Literal strings
- Expressions with CHAR or VARCHAR data types.

An expression with a CHAR data type might be a built-in or user-defined function, or one of the many other kinds of expressions available.

For more information on expressions, see “Expressions” on page 23.

A literal string is any sequence of characters enclosed in apostrophes ('single quotes'). A SQL variable of character data type can hold a string. This is a simple example of a literal string:

'This is a string.'

Special characters in strings

Represent special characters in strings by escape sequences, as follows:
• To represent an apostrophe inside a string, use two apostrophes in a row. For example:
  'John''s database'

• To represent a newline character, use a backslash followed by n (\n). For example:
  'First line:\nSecond line:'

• To represent a backslash character, use two backslashes in a row (\\). For example:
  'c:\\temp'

• Hexadecimal escape sequences can be used for any character, printable or not. A hexadecimal escape sequence is a backslash followed by an x followed by two hexadecimal digits (for example, \x6d represents the letter m). For example:
  '\x00\x01\x02\x03'

Compatibility
For compatibility with Adaptive Server Enterprise, you can set the QUOTED_IDENTIFIER database option to OFF. With this setting, you can also use double quotes to mark the beginning and end of strings. The option is ON by default.

Expressions

Syntax
expression:
  case-expression
  | constant
  | [ [ correlation-name. ] column-name [ java-ref ]
  | - expression
  | expression operator expression
  | ( expression )
  | function-name ( expression, ... )
  | if-expression
  | [ [ java-package-name. ] java-class-name java-ref
  | ( subquery )
  | variable-name [ java-ref ]

Parameters
  case-expression:
  { CASE search-condition
  ... WHEN expression THEN expression [ , ... ]
  ... [ ELSE expression ]

Reference: Building Blocks, Tables, and Procedures
Expressions

END
| CASE
... WHEN search-condition THEN expression [ , ... ]
... [ ELSE expression ]
END }

constant:
{ integer | number | 'string' | special-constant | host-variable }

special-constant:
{ CURRENT { DATE | TIME | TIMESTAMP | USER }
| LAST USER
| NULL
| SQLCODE
| SQLSTATE }

if-expression:
IF condition
... THEN expression
... [ ELSE expression ]
ENDIF

java-ref:
{ .field-name [ java-ref ]
| >> field-name [ java-ref ]
| .method-name ( [ expression ] [ , ... ] ) [ java-ref ]
| >> method-name ( [ expression ] [ , ... ] ) [ java-ref ] }

operator:
{ + | - | * | / | || | % }

Usage
Anywhere

Authorization
Must be connected to the database

Side effects
None

Description
Expressions are formed from several different kinds of element, discussed in the following sections.

Compatibility
• The IF condition is not supported in Adaptive Server Enterprise.
• Java expressions are not currently supported in Adaptive Server Enterprise.
• For other differences, see the separate descriptions of each class of expression, in the following sections.
Constants in expressions

Constants are numbers or strings. String constants are enclosed in apostrophes. An apostrophe is represented inside the string by two apostrophes in a row.

Column names in expressions

A column name is an identifier preceded by an optional correlation name. A correlation name is usually a table name. For more information on correlation names, see FROM clause in Reference: Statements and Options. If a column name has characters other than letters, digits, and underscores, the name must be surrounded by quotation marks ("""). For example, the following are valid column names:

- Employees.Surname
- City
- "StartDate"

See “Identifiers” on page 21.

Subqueries in expressions

A subquery is a SELECT statement enclosed in parentheses. The SELECT statement can contain one and only one select list item. When used as an expression, a scalar subquery is allowed to return only zero or one value;

Within the SELECT list of the top level SELECT, or in the SET clause of an UPDATE statement, you can use a scalar subquery anywhere that you can use a column name. However, the subquery cannot appear inside a conditional expression (CASE, IF, NULLIF, ARGN).

For example, the following statement returns the number of employees in each department, grouped by department name:

```sql
SELECT DepartmentName, COUNT(*), 'out of',
(SELECT COUNT(*) FROM Employees)
FROM Departments AS D, Employees AS E
WHERE D.DepartmentID = E.DepartmentID
GROUP BY DepartmentName;
```

For other uses of subqueries, see “Subqueries in search conditions” on page 35.
Expressions

SQL operators

This section describes the arithmetic, string, and bitwise operators available in Sybase IQ. For information on comparison operators, see "Search conditions" on page 33.

The normal precedence of operations applies. Expressions in parentheses are evaluated first; then multiplication and division before addition and subtraction. String concatenation occurs after addition and subtraction.

Arithmetic operators

- $expression + expression$  
  Addition. If either expression is the NULL value, the result is the NULL value.

- $expression - expression$  
  Subtraction. If either expression is the NULL value, the result is the NULL value.

- $- expression$  
  Negation. If the expression is the NULL value, the result is the NULL value.

- $expression * expression$  
  Multiplication. If either expression is the NULL value, the result is the NULL value.

- $expression / expression$  
  Division. If either expression is the NULL value or if the second expression is 0, the result is the NULL value.

- $expression % expression$  
  Modulo finds the integer remainder after a division involving two whole numbers. For example, $21 \% 11 = 10$ because $21$ divided by $11$ equals $1$ with a remainder of $10$.

String operators

- $expression || expression$  
  String concatenation (two vertical bars). If either string is the NULL value, the string is treated as the empty string for concatenation.

- $expression + expression$  
  Alternative string concatenation. When using the + concatenation operator, you must ensure the operands are explicitly set to character data types rather than relying on implicit data conversion.

The result data type of a string concatenation operator is a LONG VARCHAR. If you use string concatenation operators in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set LEFT to the correct data type and size.

See "REVERSE function [String]" on page 244 for information and usage.
Standards and compatibility

- **SQL92** The || operator is the SQL92 string concatenation operator.
- **Sybase** The + operator is supported by Adaptive Server Enterprise.

Bitwise operators

You can use the following operators on all unscaled integer data types, in both Sybase IQ and Adaptive Server Enterprise.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>EXCLUSIVE OR</td>
</tr>
<tr>
<td>~</td>
<td>NOT</td>
</tr>
</tbody>
</table>

The AND operator (&)

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 1 &amp; Bit 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The AND operator compares 2 bits. If they are both 1, the result is 1.

Bitwise OR (|)

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The OR operator compares 2 bits. If one or the other bit is 1, the result is 1.

EXCLUSIVE OR (^)

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 1 ^ Bit 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The EXCLUSIVE OR operator results in a 1 when either, but not both, of its two operands is 1.
Expressions

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 1 ^ Bit 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**NOT (~)**

The NOT operator is a unary operator that returns the inverse of its operand.

<table>
<thead>
<tr>
<th>Bit</th>
<th>~ Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Join operators**

The Transact-SQL™ outer join operators *=` and `=* are supported in Sybase IQ, in addition to the SQL92 join syntax using a table expression in the **FROM** clause.

**Compatibility**

- **Modulo** The default value is OFF for new databases.
- **String concatenation** When you are using the + concatenation operator in Sybase IQ, ensure the operands are explicitly set to strings rather than relying on implicit data conversion. For example, the following query returns the integer value 579:

  ```sql
  SELECT 123 + 456
  ```

  whereas the following query returns the character string `123456`:

  ```sql
  SELECT '123' + '456'
  ```

  You can use the CAST or CONVERT function to explicitly convert data types.

  **Note** When used with BINARY or VARBINARY data types, the + operator is concatenation, not addition.

<table>
<thead>
<tr>
<th>Bit 1 Bit 2 Bit 1 ^ Bit 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 1 01</td>
</tr>
<tr>
<td>11 01</td>
</tr>
</tbody>
</table>

The || concatenation operator is not supported by Adaptive Server Enterprise.
Operator precedence

When you are using more than one operator in an expression, Sybase recommends that you use parentheses to make the order of operation explicit, rather than relying on an identical operator precedence between Adaptive Server Enterprise and Sybase IQ.

IF expressions

The syntax of the IF expression is as follows:

```
IF condition
  THEN expression1
  [ ELSE expression2 ]
ENDIF
```

This expression returns:

- If `condition` is TRUE, the IF expression returns `expression1`.
- If `condition` is FALSE, the IF expression returns `expression2`.
- If `condition` is FALSE, and there is no `expression2`, the IF expression returns NULL.
- If condition is NULL, the IF expression returns NULL.

For more information about TRUE, FALSE, and UNKNOWN conditions, see “NULL value” on page 67 and “Search conditions” on page 33.

**IF statement is different from IF expression**

Do not confuse the syntax of the IF expression with that of the IF statement.

See IF statement in Reference: Statements and Options.

CASE expressions

The CASE expression provides conditional SQL expressions. You can use case expressions anywhere you can use an expression.

The syntax of the CASE expression is as follows:

```
CASE expression
  WHEN expression THEN expression [, …]
  [ ELSE expression ] END
```
You cannot use a subquery as a value expression in a CASE statement.

If the expression following the CASE statement is equal to the expression following the WHEN statement, then the expression following the THEN statement is returned. Otherwise, the expression following the ELSE statement is returned, if it exists.

For example, the following code uses a case expression as the second clause in a SELECT statement.

```sql
SELECT ID, 
  (CASE name
       WHEN 'Tee Shirt' THEN 'Shirt'
       WHEN 'Sweatshirt' THEN 'Shirt'
       WHEN 'Baseball Cap' THEN 'Hat'
       ELSE 'Unknown'
     END) as Type
FROM "GROUPO".Products
```

An alternative syntax is:

```sql
CASE
  WHEN search-condition THEN expression [, ...]
  [ ELSE expression ] END
```

If the search condition following the WHEN statement is satisfied, the expression following the THEN statement is returned. Otherwise the expression following the ELSE statement is returned, if it exists.

For example, the following statement uses a case expression as the third clause of a SELECT statement to associate a string with a search condition.

```sql
SELECT ID, name, 
  (CASE
      WHEN name='Tee Shirt' THEN 'Sale'
      WHEN quantity >= 50  THEN 'Big Sale'
      ELSE 'Regular price'
    END) as Type
FROM "GROUPO".Products
```

The NULLIF function provides a way to write some CASE statements in short form. The syntax for NULLIF is as follows:

```sql
NULLIF ( expression-1, expression-2 )
```

NULLIF compares the values of the two expressions. If the first expression equals the second expression, NULLIF returns NULL. If the first expression does not equal the second expression, NULLIF returns the first expression.
Compatibility of expressions

Table 2-2 and Table 2-3 describe the compatibility of expressions and constants between Adaptive Server Enterprise (ASE) and Sybase IQ. These tables are a guide only, and a marking of Both may not mean that the expression performs in an identical manner for all purposes under all circumstances. For detailed descriptions, see the Adaptive Server Enterprise documentation and the Sybase IQ documentation on the individual expression.

In Table 2-2, expr represents an expression, and op represents an operator.

**Table 2-2: Compatibility of expressions between ASE and Sybase IQ**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Supported by</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>Both</td>
</tr>
<tr>
<td>column name</td>
<td>Both</td>
</tr>
<tr>
<td>variable name</td>
<td>Both</td>
</tr>
<tr>
<td>function ( expr )</td>
<td>Both</td>
</tr>
<tr>
<td>- expr</td>
<td>Both</td>
</tr>
<tr>
<td>expr op expr</td>
<td>Both</td>
</tr>
<tr>
<td>( expr )</td>
<td>Both</td>
</tr>
<tr>
<td>( subquery )</td>
<td>Both</td>
</tr>
<tr>
<td>if-expression</td>
<td>Sybase IQ</td>
</tr>
</tbody>
</table>

**Table 2-3: Compatibility of constants between ASE and Sybase IQ**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Supported by</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>Both</td>
</tr>
<tr>
<td>number</td>
<td>Both</td>
</tr>
<tr>
<td>'string'</td>
<td>Both</td>
</tr>
<tr>
<td>special-constant</td>
<td>Both</td>
</tr>
<tr>
<td>host-variable</td>
<td>Sybase IQ</td>
</tr>
</tbody>
</table>

Default interpretation of delimited strings

By default, Adaptive Server Enterprise and Sybase IQ give different meanings to delimited strings: that is, strings enclosed in apostrophes (single quotes) and in quotation marks (double quotes).

Sybase IQ employs the SQL92 convention, that strings enclosed in apostrophes are constant expressions, and strings enclosed in quotation marks (double quotes) are delimited identifiers (names for database objects). Adaptive Server Enterprise employs the convention that strings enclosed in quotation marks are constants, whereas delimited identifiers are not allowed by default and are treated as strings.
The quoted_identifier option

Both Adaptive Server Enterprise and Sybase IQ provide a quoted_identifier option that allows the interpretation of delimited strings to be changed. By default, the quoted_identifier option is set to OFF in Adaptive Server Enterprise, and to ON in Sybase IQ.

You cannot use SQL reserved words as identifiers if the quoted_identifier option is off.

For a complete list of reserved words, see Table 2-1 on page 19.

Setting the option

Although the Transact-SQL SET statement is not supported for most Adaptive Server Enterprise connection options, SET is supported for the quoted_identifier option.

The following statement in either Sybase IQ or Adaptive Server Enterprise changes the setting of the quoted_identifier option to ON:

    SET quoted_identifier ON

With the quoted_identifier option set to ON, Adaptive Server Enterprise allows table, view, and column names to be delimited by quotes. Other object names cannot be delimited in Adaptive Server Enterprise.

The following statement in Sybase IQ or Adaptive Server Enterprise changes the setting of the quoted_identifier option to OFF:

    SET quoted_identifier OFF

You can choose to use either the SQL92 or the default Transact-SQL convention in both Adaptive Server Enterprise and Sybase IQ as long as the quoted_identifier option is set to the same value in each DBMS.

Examples

If you operate with the quoted_identifier option ON (the default Sybase IQ setting), the following statements involving the SQL keyword user are valid for both types of DBMS.

    CREATE TABLE "user" (  
        col1 char(5)  
    ) ;
    INSERT "user" ( col1 ) VALUES ( 'abcde' ) ;

If you operate with the quoted_identifier option OFF (the default Adaptive Server Enterprise setting), the following statements are valid for both types of DBMS.

    SELECT *
    FROM Employees
WHERE Surname = "Chin"

Search conditions

Function
To specify a search condition for a WHERE clause, a HAVING clause, a CHECK clause, a JOIN clause, or an IF expression.

Syntax
{ expression compare expression
| expression compare { ANY | SOME | ALL } ( subquery )
| expression IS | NOT | NULL
| expression | NOT | BETWEEN expression AND expression
| expression | NOT | LIKE expression [ ESCAPE expression ]
| expression | NOT | IN ( { expression | subquery | ...
value-expr1, value-expr2 [, value-expr3 ] ... } )
| column-name | NOT | CONTAINS ( ... word1 [, word2 ] [, word3 ] ... )
| EXISTS ( subquery )
| NOT condition
| condition AND condition
| condition OR condition
| ( condition )
| ( condition , estimate )
| condition IS | NOT | { TRUE | FALSE | UNKNOWN }

Parameters
compare:
{ = | > | < | >= | <= | <> | != | !< | !> }

Usage
Anywhere

Authorization
Must be connected to the database

Example
For example, the following query retrieves the names and birth years of the oldest employees:

```
SELECT Surname, BirthDate
FROM Employees
WHERE BirthDate <= ALL (SELECT BirthDate FROM Employees);
```

The subqueries that provide comparison values for quantified comparison predicates might retrieve multiple rows but can have only one column.

Side effects
None

See also
“Expressions” on page 23.

Description
Conditions are used to choose a subset of the rows from a table, or in a control statement such as an IF statement to determine control of flow.
**Search conditions**

SQL conditions do not follow Boolean logic, where conditions are either true or false. In SQL, every condition evaluates as one of TRUE, FALSE, or UNKNOWN. This is called three-valued logic. The result of a comparison is UNKNOWN if either value being compared is the NULL value. For tables showing how logical operators combine in three-valued logic, see “Three-valued logic” on page 46.

Rows satisfy a search condition if and only if the result of the condition is TRUE. Rows for which the condition is UNKNOWN do not satisfy the search condition. For more information, see “NULL value” on page 67.

Subqueries form an important class of expression that is used in many search conditions. For more information, see “Subqueries in search conditions” on page 35.

The different types of search conditions are discussed in the following sections.

**Comparison conditions**

The syntax for comparison conditions is as follows:

```
expression compare expression
```

where `compare` is a comparison operator. Table 2-4 lists the comparison operators available in Sybase IQ.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>!&gt;</td>
<td>Not greater than</td>
</tr>
<tr>
<td>&lt;!</td>
<td>Not less than</td>
</tr>
</tbody>
</table>

Example

For example, the following query retrieves the names and birth years of the oldest employees:

```sql
SELECT Surname, BirthDate
FROM Employees
WHERE Surname <= ALL (SELECT MIN(BirthDate) FROM Employees)
```
The subqueries that provide comparison values for quantified comparison predicates, as in the preceding example, might retrieve multiple rows but can only have one column.

**Note** All string comparisons are:

- Case-sensitive if the database was created as case respect (the default)
- Case-insensitive if the database was created as case ignore

See the “Usage” section of the CREATE DATABASE statement in *Reference: Statements and Options* for more details on the results of comparisons in a case-insensitive database.

**Compatibility**

- **Trailing blanks** Any trailing blanks in character data are ignored for comparison purposes by Adaptive Server Enterprise. The behavior of Sybase IQ when comparing strings is controlled by the Ignore Trailing Blanks in String Comparisons database creation option.
- **Case sensitivity** By default, Sybase IQ databases, like Adaptive Server Enterprise databases, are created as case-sensitive. Comparisons are carried out with the same attention to case as the database they are operating on. You can control the case sensitivity of Sybase IQ databases when creating the database.

**Subqueries in search conditions**

A subquery is a SELECT statement enclosed in parentheses. Such a SELECT statement must contain one and only one select list item.

A column can be compared to a subquery in a comparison condition (for example, >,<, or !={}) as long as the subquery returns no more than one row. If the subquery (which must have one column) returns one row, the value of that row is compared to the expression. If a subquery returns no rows, its value is NULL.

Subqueries that return exactly one column and any number of rows can be used in IN conditions, ANY conditions, ALL conditions, or EXISTS conditions. These conditions are discussed in the following sections.

Sybase IQ supports UNION only in uncorrelated subquery predicates, not in scalar value subqueries or correlated subquery predicates.
Subqueries cannot be used inside a CONTAINS or LIKE predicate.

Sybase IQ does not support multiple subqueries in a single OR clause. For example, the following query has two subqueries joined by an OR:

```sql
CREATE VARIABLE @ln int;
SELECT @ln = 1; select count(*) FROM lineitem
WHERE l_shipdate IN (select l_shipdate FROM lineitem
WHERE l_orderkey IN (2,4,6))
OR l_shipdate IN (select l_shipdate FROM lineitem WHERE
l_orderkey IN (1,3,5))
OR l_linenumber = @ln;
```

Similar subqueries joined by AND and BETWEEN are allowed.

See “Comparison conditions” on page 34.

**Disjunction of subquery predicates**

The SQL89 standard allows for several forms of subquery predicates. Each subquery can appear within the WHERE or HAVING clause with other predicates, and can be combined using the AND or OR operators. Sybase IQ supports these subqueries, which can be correlated (contain references to a table that appears in the outer query and cannot be evaluated independently) and uncorrelated (do not contain references to remote tables).

The forms of subquery predicates include:

- **Unquantified comparison predicates:**
  ```sql
  <scalar-expression> <comparison-operator> <subquery>
  ```
  The comparison operator is: =, <>, >, >=, <, or <=

  Unquantified comparison subqueries return exactly one value. If the subquery returns more than one value, an error message appears. This type of query is also called a scalar subquery predicate.

- **IN predicates:**
  ```sql
  <scalar-expression> [NOT] IN <subquery>
  ```
  The IN subquery predicate returns a list of values or a single value. This type is also called a quantified subquery predicate.

- **Existence predicates:**
[NOT] EXISTS <subquery>

The EXISTS predicate represents the existence of the subquery. The expression EXISTS <subquery> evaluates to true only if the subquery result is not empty. The EXISTS predicate does not compare results with any column or expressions in the outer query block, and is typically used with correlated subqueries.

- Quantified comparison predicates:

  <scalar-expression> <comparison-operator> [ANY | ALL] <subquery>

A quantified comparison predicate compares one or a collection of values returned from a subquery.

The types of queries you can run include:

- Disjunction of uncorrelated scalar subqueries or IN subqueries that cannot be executed vertically within the WHERE or HAVING clause
- Disjunction of correlated/uncorrelated EXISTS subqueries within the WHERE or HAVING clause
- Disjunction of arbitrary correlated/uncorrelated scalar subqueries, IN or EXISTS subqueries, or quantified comparison subqueries within the WHERE or HAVING clause
- Arbitrary uncorrelated/correlated subquery predicates combined with AND/OR (conjunct/disjunct) and simple predicates or subquery predicates
- Conjunction/disjunction of subquery predicates on top of a view/derived table
- Disjunction of subquery predicates in UPDATE, DELETE, and SELECT INTO statements

The SUBQUERY_CACHING_PREFERENCE option lets experienced DBAs choose which subquery caching method to use. See SUBQUERY_CACHING_PREFERENCE option in Reference: Statements and Options.

Examples

Example 1  Disjunction of uncorrelated EXISTS and IN subqueries:

```
SELECT COUNT(*)
FROM supplier
WHERE s_suppkey IN (SELECT MAX(l_suppkey)
                     FROM lineitem
                     GROUP BY l_linenumber)
```
Search conditions

OR EXISTS (SELECT p_brand
            FROM part
            WHERE p_brand = 'Brand#43');

Example 2 Disjunction of uncorrelated EXISTS subqueries:

SELECT COUNT(*)
FROM supplier
WHERE EXISTS (SELECT l_suppkey
               FROM lineitem
               WHERE l_suppkey = 12345)
OR EXISTS (SELECT p_brand
            FROM part
            WHERE p_brand = 'Brand#43');

Example 3 Disjunction of uncorrelated scalar or IN subquery predicates:

SELECT COUNT(*)
FROM supplier
WHERE s_acctbal*10 > (SELECT MAX(o_totalprice)
                      FROM orders
                      WHERE o_custkey = 12345)
OR substring(s_name, 1, 6) IN (SELECT c_name
                               FROM Customers
                               WHERE c_nationkey = 10);

Example 4 Disjunction of correlated/uncorrelated quantified comparison subqueries:

SELECT COUNT(*)
FROM lineitem
WHERE l_suppkey > ANY (SELECT MAX(s_suppkey)
                         FROM supplier
                         WHERE s_acctbal > 100
                         GROUP BY s_nationkey)
OR l_partkey >= ANY (SELECT MAX(p_partkey)
                      FROM part
                      GROUP BY p_mfgr);

Example 5 Disjunction of any correlated subquery predicates:

SELECT COUNT(*)
FROM supplier S
WHERE EXISTS (SELECT l_suppkey
               FROM lineitem
               WHERE l_suppkey = S.s_suppkey)
OR EXISTS (SELECT p_brand FROM part
            WHERE p_brand = 'Brand#43'
            AND p_partkey > S.s_suppkey);
Before support for disjunction of subqueries, users were required to write queries in two parts, and then use UNION to merge the final results.

The following query illustrates a merged query that gets the same results as the query in “Example 5” on page 38. Performance of the merged query is suboptimal because it scans the supplier table twice and then merges the results from each UNION to return the final result.

```
SELECT COUNT(*)
FROM (SELECT s_suppkey FROM supplier S
       WHERE EXISTS (SELECT l_suppkey
                      FROM lineitem
                      WHERE l_suppkey = S.s_suppkey)
       UNION
       SELECT s_suppkey FROM supplier S
       WHERE EXISTS (SELECT p_brand
                      FROM part
                      WHERE p_brand = 'Brand#43'
                      AND p_partkey > S.s_suppkey)) as UD;
```

### ALL or ANY conditions

The syntax for ANY conditions is:

```
expression compare ANY ( subquery )
```

where `compare` is a comparison operator.

For example, an ANY condition with an equality operator is TRUE if `expression` is equal to any of the values in the result of the subquery, and FALSE if the expression is not NULL and does not equal any of the columns of the subquery:

```
expression = ANY ( subquery )
```

The ANY condition is UNKNOWN if `expression` is the NULL value, unless the result of the subquery has no rows, in which case the condition is always FALSE.

You can use the keyword SOME instead of ANY.

The syntax for ALL conditions is:

```
expression compare ALL ( subquery )
```

where `compare` is a comparison operator.
Search conditions

Restrictions
If there is more than one expression on either side of a quantified comparison predicate, an error message is returned. For example:

Subquery allowed only one select list item

Queries of this type can always be expressed in terms of IN subqueries or scalar subqueries using MIN and MAX set functions.

Compatibility
ANY and ALL subqueries are compatible between Adaptive Server Enterprise and Sybase IQ. Only Sybase IQ supports SOME as a synonym for ANY.

BETWEEN conditions

The syntax for BETWEEN conditions is as follows:

```
expr [ NOT ] BETWEEN start-expr AND end-expr
```

The BETWEEN condition can evaluate as TRUE, FALSE, or UNKNOWN. Without the NOT keyword, the condition evaluates as TRUE if `expr` is between `start-expr` and `end-expr`. The NOT keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

The BETWEEN condition is equivalent to a combination of two inequalities:

```
expr >= start-expr AND expr <= end-expr
```

A BETWEEN predicate is of the form “A between B and C.” Either “B” or “C” or both “B” and “C” can be subqueries. “A” must be a value expression or column.

Compatibility
The BETWEEN condition is compatible between Sybase IQ and Adaptive Server Enterprise.

LIKE conditions

The syntax for LIKE conditions is:

```
expression [ NOT ] LIKE pattern [ ESCAPE escape-expr ]
```

The LIKE condition can evaluate as TRUE, FALSE, or UNKNOWN. You can use LIKE only on string data.

You cannot use subqueries inside a LIKE predicate.

LIKE predicates that start with characters other than wildcard characters may execute faster if an HG or LF index is available.
Certain LIKE predicates execute faster, if a WD index is available.

Without the NOT keyword, the condition evaluates as TRUE if expression matches the pattern. If either expression or pattern is the NULL value, this condition is UNKNOWN. The NOT keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

The pattern might contain any number of wildcard characters. The wildcard characters are:

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>(underscore)</td>
<td>Any one character</td>
</tr>
<tr>
<td>% (percent)</td>
<td>Any string of zero or more characters</td>
</tr>
<tr>
<td>[ ]</td>
<td>Any single character in the specified range or set</td>
</tr>
<tr>
<td>[^]</td>
<td>Any single character not in the specified range or set</td>
</tr>
</tbody>
</table>

All other characters must match exactly.

For example, the search condition:

```
name LIKE 'a%b_'
```

is TRUE for any row where name starts with the letter a and has the letter b as its second-to-last character.

If you specify an escape-expr, it must evaluate to a single character. The character can precede a percent, an underscore, a left square bracket, or another escape character in the pattern to prevent the special character from having its special meaning. When escaped in this manner, a percent matches a percent, and an underscore matches an underscore.

All patterns of 126 characters or less are supported. Patterns of length greater than 254 characters are not supported. Some patterns of length between 127 and 254 characters are supported, depending on the contents of the pattern.

### Searching for one of a set of characters

You can specify a set of characters to look for by listing the characters inside square brackets. For example, the following condition finds the strings smith and smyth:

```
LIKE 'sm[iy]th'
```

### Searching for one of a range of characters

Specify a range of characters to look for by listing the ends of the range inside square brackets, separated by a hyphen. For example, the following condition finds the strings bough and rough, but not tough:

```
LIKE '^[a-r]ough'
```
Search conditions

The range of characters [a-z] is interpreted as “greater than or equal to a, and less than or equal to z,” where the greater than and less than operations are carried out within the collation of the database. For information on ordering of characters within a collation, see Chapter 11, “International Languages and Character Sets” in the System Administration Guide: Volume 1.

The lower end of the range must precede the higher end of the range. For example, a LIKE condition containing the expression [z-a] returns no rows, because no character matches the [z-a] range.

Unless the database is created as case-sensitive, the range of characters is case-insensitive. For example, the following condition finds the strings Bough, rough, and TOUGH:

LIKE '^[a-z]ough'

If the database is created as a case-sensitive database, the search condition is case-sensitive also.

Combining searches for ranges and sets

You can combine ranges and sets within square brackets. For example, the following condition finds the strings bough, rough, and tough:

LIKE '[a-rt]ough'

The bracket [a-mpqs-z] is interpreted as “exactly one character that is either in the range a to m inclusive, or is p, or is q, or is in the range s to z inclusive.”

Searching for one character not in a range

Use the caret character (^) to specify a range of characters that is excluded from a search. For example, the following condition finds the string tough, but not the strings rough, or bough:

LIKE '^[^a-r]ough'

The caret negates the entire contents of the brackets. For example, the bracket [^a-mpqs-z] is interpreted as “exactly one character that is not in the range a to m inclusive, is not p, is not q, and is not in the range s to z inclusive.”

Special cases of ranges and sets

Any single character in square brackets indicates that character. For example, [a] matches just the character a. [^] matches just the caret character, [%] matches only the percent character (the percent character does not act as a wildcard character in this context), and [_] matches just the underscore character. Also, [\] matches only the character ".

Other special cases are:

- The expression [a-] matches either of the characters a or .
- The expression [\] is never matched and always returns no rows.
• The expressions [ or \textit{abp-q} are ill-formed expressions, and give syntax errors.
• You cannot use wildcard characters inside square brackets. The expression \[a\%b\] finds one of \textit{a}, \textit{\%}, or \textit{b}.
• You cannot use the caret character to negate ranges except as the first character in the bracket. The expression \[a^b\] finds one of \textit{a}, \^, or \textit{b}.

\textbf{Compatibility}

The \texttt{ESCAPE} clause is supported by Sybase IQ only.

\textbf{IN conditions}

The syntax for \texttt{IN} conditions is:

\begin{verbatim}
{ expression \{ NOT \ [ NOT ] IN ( subquery )
| expression \{ NOT \ [ NOT ] IN ( expression )
| expression \{ NOT \ [ NOT ] IN ( value-expr1 , value-expr2
[ , value-expr3 ] ... )
\}
\end{verbatim}

Without the \texttt{NOT} keyword, the \texttt{IN} condition is TRUE if \textit{expression} equals any of the listed values, UNKNOWN if \textit{expression} is the NULL value, and FALSE otherwise. The \texttt{NOT} keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

The maximum number of values allowed in an \texttt{IN} condition list is 250,000.

\textbf{Compatibility}

\texttt{IN} conditions are compatible between Adaptive Server Enterprise and Sybase IQ.

\textbf{CONTAINS conditions}

The syntax for \texttt{CONTAINS} conditions is as follows:

\begin{verbatim}
{ column-name \{ NOT \ CONTAINS ( ( word1 [ , word2 ] [ , word3 ] ... )
\}
\end{verbatim}

The \textit{column-name} must be a \texttt{CHAR}, \texttt{VARCHAR}, or \texttt{LONG VARCHAR (CLOB)} column in a base table, and must have a \texttt{WD} index. The \textit{word1}, \textit{word2} and \textit{word3} expressions must be string constants no longer than 255 bytes, each containing exactly one word. The length of that word cannot exceed the maximum permitted word length of the word index of the column.
Search conditions

Without the NOT keyword, the CONTAINS condition is TRUE if column-name contains each of the words, UNKNOWN if column-name is the NULL value, and FALSE otherwise. The NOT keyword reverses these values but leaves UNKNOWN unchanged.

For example, this search condition:

\[
\text{varchar}_\text{col} \text{ CONTAINS} (\text{‘cat’, ‘mat’})
\]

is TRUE if the value of \text{varchar}_\text{col} is The cat is on the mat. If the value of \text{varchar}_\text{col} is The cat chased the mouse, this condition is FALSE.

When Sybase IQ executes a statement containing both LIKE and CONTAINS, the CONTAINS condition takes precedence.

Avoid using the CONTAINS predicate in a view that has a user-defined function, because the CONTAINS criteria are ignored. Use the LIKE predicate with wildcards instead, or issue the query outside of a view.

EXISTS conditions

The syntax for EXISTS conditions is as follows:

\[
\text{EXISTS}( \text{subquery})
\]

The EXISTS condition is TRUE if the subquery result contains at least one row, and FALSE if the subquery result does not contain any rows. The EXISTS condition cannot be UNKNOWN.

Compatibility

The EXISTS condition is compatible between Adaptive Server Enterprise and Sybase IQ.

IS NULL conditions

The syntax for IS NULL conditions is:

\[
\text{expression IS} \begin{cases} \text{[ NOT] NULL} \end{cases}
\]

Without the NOT keyword, the IS NULL condition is TRUE if the expression is the NULL value, and FALSE otherwise. The NOT keyword reverses the meaning of the condition.

Compatibility

The IS NULL condition is compatible between Adaptive Server Enterprise and Sybase IQ.
Conditions with logical operators

Search conditions can be combined using AND, OR, and NOT.

Conditions are combined using AND as follows:

```
condition1 AND condition2
```

If both conditions are TRUE, the combined condition is TRUE. If either condition is FALSE, the combined condition is FALSE. If otherwise, the combined condition is UNKNOWN.

Conditions are combined using OR as follows:

```
condition1 OR condition2
```

If both conditions are TRUE, the combined condition is TRUE. If either condition is FALSE, the combined condition is FALSE. If otherwise, the combined condition is UNKNOWN. There is no guaranteed order as to which condition, condition1 or condition2, is evaluated first.

See “Disjunction of subquery predicates” on page 36.

Compatibility

The AND and OR operators are compatible between Sybase IQ and Adaptive Server Enterprise.

NOT conditions

The syntax for NOT conditions is:

```
NOT condition1
```

The NOT condition is TRUE if condition1 is FALSE, FALSE if condition1 is TRUE, and UNKNOWN if condition1 is UNKNOWN.

Truth value conditions

The syntax for truth value conditions is:

```
IS [ NOT ] truth-value
```

Without the NOT keyword, the condition is TRUE if the condition evaluates to the supplied truth-value, which must be one of TRUE, FALSE, or UNKNOWN. Otherwise, the value is FALSE. The NOT keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

Compatibility

Truth-valued conditions are supported by Sybase IQ only.
Three-valued logic

The following tables show how the AND, OR, NOT, and IS logical operators of SQL work in three-valued logic.

<table>
<thead>
<tr>
<th>AND operator</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR operator</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT operator</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IS operator</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

User-supplied condition hints

The Sybase IQ query optimizer uses information from available indexes to select an appropriate strategy for executing a query. For each condition in the query, the optimizer decides whether the condition can be executed using indexes, and if so, the optimizer chooses which index and in what order with respect to the other conditions on that table. The most important factor in these decisions is the selectivity of the condition; that is, the fraction of the table’s rows that satisfy that condition.
The optimizer normally decides without user intervention, and it generally makes optimal decisions. In some situations, however, the optimizer might not be able to accurately determine the selectivity of a condition before it has been executed. These situations normally occur only where either the condition is on a column with no appropriate index available, or where the condition involves some arithmetic or function expression and is, therefore, too complex for the optimizer to accurately estimate.

If you have a query that is run frequently, then you may want to experiment to see whether you can improve the performance of that query by supplying the optimizer with additional information to aid it in selecting the optimal execution strategy.

**User-supplied condition selectivity**

The simplest form of condition hint is to supply a selectivity value that will be used instead of the value the optimizer would have computed.

Selectivity hints are supplied within the text of the query by wrapping the condition within parentheses. Then within the parentheses, after the condition, you add a comma and a numeric value to be used as the selectivity.

This selectivity value is expressed as a percentage of the table’s rows, which satisfy the condition. Possible numeric values for selectivity thus range from 100.0 to 0.0.

**Note**  In query plans, selectivity is expressed as a fraction instead of as a percentage; so a user-supplied selectivity of 35.5 appears in that query’s plan as a selectivity of 0.355000.

**Examples**

- The following query provides an estimate that one and one half percent of the ship_date values are earlier than 1994/06/30:

  ```sql
  SELECT  ShipDate
  FROM  SalesOrderItems
  WHERE ( ShipDate < '2001/06/30', 1.5 )
  ORDER BY ShipDate DESC
  ```

- The following query estimates that half a percent of the rows satisfy the condition:

  ```sql
  SELECT *
  FROM Customers c, SalesOrders o
  WHERE (o.SalesRepresentative > 1000.0, 0.5)
      AND c.ID = o.customerID
  ```
Search conditions

Fractional percentages enable more precise user estimates to be specified and can be particularly important for large tables.

Compatibility

SQL Anywhere Studio supports user-supplied selectivity estimates.

Adaptive Server Enterprise does not support user-supplied selectivity estimates.

User-supplied condition hint strings

In addition to supporting user-supplied selectivity estimates, Sybase IQ also lets users supply additional hint information to the optimizer through a condition hint string. These per-condition hint strings let users specify additional execution preferences for a condition, which the optimizer follows, if possible. These preferences include which index to use for the condition, the selectivity of the condition, the phase of execution when the condition is executed, and the usefulness of the condition, which affects its ordering among the set of conditions executed within one phase of execution.

Condition hint strings, like the user-supplied selectivity estimates, are supplied within the text of the query by wrapping the condition within parentheses. Then within the parentheses and after the condition, you add a comma and a supply a quoted string containing the desired hints. Within that quoted string each hint appears as a hint type identifier, followed by a colon and the value for that hint type. Multiple hints within the same hint string are separated from each other by a comma, and multiple hints can appear in any order. White space is allowed between any of two elements within a hint string.

There are four different supported hint types:

- Selectivity hints, which are equivalent to the user-supplied selectivity estimates
- Index preference hints
- Execution phase hints
- Usefulness hints

Selectivity hints

The first hint type that can appear within a hint string is a selectivity hint. A selectivity hint is identified by a hint type identifier of either “S” or “s”. Like user-supplied selectivity estimates, the selectivity value is always expressed as a percentage of the table’s rows, which satisfy the condition.
Example

The following example is exactly equivalent to the second example in “User-supplied condition selectivity” on page 47.

```
SELECT *
FROM Customers c, SalesOrders o
WHERE (o.SalesRepresentative > 1000.0, 's: 0.5)
AND c.ID = o.CustomerID
```

Index preference hints

The next supported hint type is an index preference hint, which is identified by a hint type identifier of either “I” or “i”. The value for an index preference hint can be any integer between -10 and 10. The meaning of each positive integer value is to prefer a specific index type, while negative values indicate that the specific index type is to be avoided.

The effect of an index preference hint is the same as that of the `INDEX_PREFERENCE` option, except that the preference applies only to the condition it is associated with rather than all conditions within the query. An index preference can only affect the execution of a condition if the specified index type exists on that column and that index type is valid for use when evaluating the associated condition; not all index types are valid for use with all conditions. See “INDEX_PREFERENCE option,” in Chapter 2, “Database Options,” in Reference: Statements and Options for the specific meanings of integers between -10 and 10.

Example

The following example specifies a 3 percent selectivity and indicates that, if possible, the condition should be evaluated using an HG index:

```
SELECT *
FROM Customers c, SalesOrders o
WHERE (o.SalesRepresentative > 1000.0, 'S:3.00, I:+2')
AND c.ID = o.CustomerID
```

The next example specifies a 37.5 percent selectivity and indicates that if possible the condition should not be evaluated using an HG index:

```
SELECT *
FROM Customers c, SalesOrders o
WHERE (o.SalesRepresentative > 1000.0, 'i:-2, s:37.500')
AND c.ID = o.CustomerID
```

Execution phase hints

The third supported hint type is the execution phase hint, which is identified with a hint type identifier of either “E” or “e.”
Within the Sybase IQ query engine, there are distinct phases of execution where conditions can be evaluated: invariant, delayed, bound, and horizontal.

By default, the optimizer chooses to evaluate each condition within the earliest phase of execution where all the information needed to evaluate that condition is available. Every condition, therefore, has a default execution phase where it is evaluated.

Because no condition can be evaluated before the information it needs is available, the execution phase hint can only be used to delay the execution of a condition to a phase after its default phase. It cannot be used to force a condition to be evaluated within any phase earlier than its default phase.

The four phases of condition execution from earliest to latest are described as follows:

**Invariant**  A condition that refers to only one column (or two columns from the same table) and that can be evaluated using an index is generally referred to as a simple invariant condition. Simple invariant condition are normally evaluated early within the optimization process.

This means that the number of rows satisfying all of those invariant conditions is available to guide the optimizer’s decisions on the best join order and join algorithms to use. Because this is the earliest phase of execution, a user can never force a condition into this phase, but conditions can be forced out of this phase into later phases.

**Delayed**  Some conditions cannot be evaluated until some other part of a query has been executed. These delayed conditions are evaluated once when the query node to which they are attached is first fetched. These conditions fall into two categories, uncorrelated subquery conditions and IN or PROBABLY_IN pushdown join conditions created by the optimizer.

**Bound**  Some conditions must be evaluated multiple times. These conditions generally fall into two categories: conditions containing outer references within a correlated subquery, and pushdown equality join conditions created by the optimizer. The outer reference conditions, for example, are reevaluated each time the outer reference value changes during the query’s execution.

**Horizontal**  Some conditions, such as those which contain more than two columns from a table, must be evaluated one row at a time, rather than by using an index.

An execution phase hint accepts a values that identifies in which execution phase the user wants the condition to be evaluated. Each value is a case-insensitive single character:

- D – Delayed
• B – Bound
• H – Horizontal

Example
The following example shows a condition hint string which indicates that the condition should be moved into the “Delayed” phase of execution, and it indicates that if possible the condition should be evaluated using an LF index:

```sql
SELECT *
FROM Customers c, SalesOrders o
WHERE (o.SalesRepresentative > 10000.0, 'E:D, I:1')
    AND c.id = o.CustomerID
```

Usefulness hints
The final supported hint type is the usefulness hint, which is identified by a hint type identifier of either “U” or “u”. The value for a usefulness hint can be any numeric value between 0.0 and 10.0. Within the optimizer a usefulness value is computed for every condition, and the usefulness value is then used to determine the order of evaluation among the set of conditions to be evaluated within the same phase of execution. The higher the usefulness value, the earlier it appears in the order of evaluation. Supplying a usefulness hint lets users place a condition at a particular point within the order of evaluation, but it cannot change the execution phase within which the condition is evaluated.

Example
The following example shows a condition hint string which indicates that the condition should be moved into the “Delayed” phase of execution, and that its usefulness should be set to 3.25 within that “Delayed” phase.

```sql
SELECT *
FROM Customers c, SalesOrders o
WHERE (co.SalesRepresentative > 10000.0, 'U: 3.25,  E: D')
    AND c.id = o.CustomerID
```

Compatibility
SQL Anywhere Studio does not support user-supplied condition hint strings. Adaptive Server Enterprise does not support user-supplied condition hint strings.

User-supplied hints on join equality conditions
Users can specify a join algorithm preference that does not affect every join in the query.
Simple equality join predicates can be tagged with a predicate hint that allows a join preference to be specified for just that one join. If the same join has more than one join condition with a local join preference, and if those hints are not the same value, then all local preferences are ignored for that join. Local join preferences do not affect the join order chosen by the optimizer.

The following example requests a hash join:

\[ \text{AND} \ (T.X = 10 * R.x, \ 'J:4') \]

**Guidelines for usage of user-supplied condition hints**

Condition hints are generally appropriate only within frequently run queries. Only advanced users should experiment with condition hints. The optimizer generally makes optimal decisions, except where it cannot infer accurate information about a condition from the available indexes.

The optimizer often rewrites or simplifies the original conditions, and it also infers new conditions from the original conditions. Condition hints are not carried through new to conditions inferred by the optimizer, nor are they carried through to simplified conditions.
Special values

Special values can be used in expressions, and as column defaults when creating tables.

**CURRENT DATABASE special value**

Function: CURRENT DATABASE returns the name of the current database.

Data type: STRING

See also: “Expressions” on page 23

**CURRENT DATE special value**

Function: The current year, month and day.

Data type: DATE

See also: “Expressions” on page 23

“Date and time data types” on page 82

**CURRENT PUBLISHER special value**

Function: CURRENT PUBLISHER returns a string that contains the publisher user ID of the database for SQL Remote replications.

Data type: STRING

Current PUBLISHER can be used as a default value in columns with character data types.

See also: “Expressions” on page 23

**CURRENT TIME special value**

Function: The current hour, minute, second, and fraction of a second.

Data type: TIME
Special values

Description: The fraction of a second is stored to 6 decimal places, but the accuracy of the current time is limited by the accuracy of the system clock.

See also: “Expressions” on page 23
“Date and time data types” on page 82

CURRENT TIMESTAMP special value

Function: Combines CURRENT DATE and CURRENT TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second and fraction of a second. As with CURRENT TIME, the accuracy of the fraction of a second is limited by the system clock.

CURRENT TIMESTAMP defaults to 3 digits.

Data type: TIMESTAMP

See also: “Expressions” on page 23
“Date and time data types” on page 82

CURRENT USER special value

Function: CURRENT USER returns a string that contains the user ID of the current connection.

Data type: STRING

CURRENT USER can be used as a default value in columns with character data types.

Description: On UPDATE, columns with a default value of CURRENT USER are not changed.

See also: “Expressions” on page 23

LAST USER special value

Function: The name of the user who last modified the row.

Data type: STRING
LAST USER can be used as a default value in columns with character data types.

Description

On INSERT and LOAD, this constant has the same effect as CURRENT USER. On UPDATE, if a column with a default value of LAST USER is not explicitly modified, it is changed to the name of the current user.

When combined with the DEFAULT TIMESTAMP, a default value of LAST USER can be used to record (in separate columns) both the user and the date and time a row was last changed.

See also

“CURRENT USER special value” on page 54
“CURRENT TIMESTAMP special value” on page 54

CREATE TABLE statement in Reference: Statements and Options

SQLCODE special value

Function

Current SQLCODE value.

DATA TYPE

STRING

DESCRIPTION

The SQLCODE value is set after each statement. You can check the SQLCODE to see whether or not the statement succeeded.

See also

“Expressions” on page 23


SQLSTATE special value

Function

Current SQLSTATE value.

Data type

STRING

Description

The SQLSTATE value is set after each statement. You can check the SQLSTATE to see whether or not the statement succeeded.

See also

“Expressions” on page 23

TIMESTAMP special value

Function
TIMESTAMP indicates when each row in the table was last modified.

Data type
TIMESTAMP

Description
When a column is declared with DEFAULT TIMESTAMP, a default value is provided for insert and load operations. The value is updated with the current date and time whenever the row is updated.

On INSERT and LOAD, DEFAULT TIMESTAMP has the same effect as CURRENT TIMESTAMP. On UPDATE, if a column with a default value of TIMESTAMP is not explicitly modified, the value of the column is changed to the current date and time.

Note
Sybase IQ does not support DEFAULT values of UTC TIMESTAMP or CURRENT UTC TIMESTAMP, nor does IQ support the database option DEFAULT_TIMESTAMP_INCREMENT. Sybase IQ generates an error every time an attempt is made to insert or update the DEFAULT value of a column of type UTC TIMESTAMP or CURRENT UTC TIMESTAMP.

See also
“Date and time data types” on page 82

USER special value

Function
USER returns a string that contains the user ID of the current connection.

Data type
STRING

Description
USER can be used as a default value in columns with character data types.

On UPDATE, columns with a default value of USER are not changed.

See also
“Expressions” on page 23
“CURRENT USER special value” on page 54
“LAST USER special value” on page 54

Variables

Sybase IQ supports three levels of variables:
• **Local variables** These are defined inside a compound statement in a procedure or batch using the DECLARE statement. They exist only inside the compound statement.

• **Connection-level variables** These are defined with a CREATE VARIABLE statement. They belong to the current connection, and disappear when you disconnect from the database or when you use the DROP VARIABLE statement.

• **Global variables** These are variables that have system-supplied values.

Local and connection-level variables are declared by the user, and can be used in procedures or in batches of SQL statements to hold information. Global variables are system-supplied variables that provide system-supplied values. All global variables have names beginning with two @ signs. For example, the global variable @@version has a value that is the current version number of the database server. Users cannot define global variables.

### Local variables

Local variables are declared using the DECLARE statement, which can be used only within a compound statement (that is, bracketed by the BEGIN and END keywords). The variable is initially set as NULL. You can set the value of the variable using the SET statement, or you can assign the value using a SELECT statement with an INTO clause.

The syntax of the DECLARE statement is as follows:

```sql
DECLARE variable-name data-type
```

You can pass local variables as arguments to procedures, as long as the procedure is called from within the compound statement.

**Examples**

- The following batch illustrates the use of local variables:

```sql
BEGIN
  DECLARE local_var INT ;
  SET local_var = 10 ;
  MESSAGE 'local_var = ', local_var ;
END
```

Running this batch from ISQL displays this message on the server window:

```
local_var = 10
```
Variables

- The variable `local_var` does not exist outside the compound statement in which it is declared. The following batch is invalid, and displays a column not found error:

  ```sql
  -- This batch is invalid.
  BEGIN
  DECLARE local_var INT ;
  SET local_var = 10 ;
  MESSAGE 'local_var = ', local_var ;
  END;
  MESSAGE 'local_var = ', local_var ;
  ```

- The following example illustrates the use of `SELECT` with an `INTO` clause to set the value of a local variable:

  ```sql
  BEGIN
  DECLARE local_var INT ;
  SELECT 10 INTO local_var ;
  MESSAGE 'local_var = ', local_var ;
  END
  ```

  Running this batch from ISQL displays this message on the server window:

  `local_var = 10`

Compatibility

- **Names** Adaptive Server Enterprise and Sybase IQ both support local variables. In Adaptive Server Enterprise, all variables must be prefixed with an `@` sign. In Sybase IQ, the `@` prefix is optional. To write compatible SQL, ensure all your variables have the `@` prefix.

- **Scope** The scope of local variables differs between Sybase IQ and Adaptive Server Enterprise. Sybase IQ supports the use of the `DECLARE` statement to declare local variables within a batch. However, if the `DECLARE` is executed within a compound statement, the scope is limited to the compound statement.

- **Declaration** Only one variable can be declared for each `DECLARE` statement in Sybase IQ. In Adaptive Server Enterprise, more than one variable can be declared in a single statement.
Connection-level variables

Connection-level variables are declared with the `CREATE VARIABLE` statement. The `CREATE VARIABLE` statement can be used anywhere except inside compound statements. Connection-level variables can be passed as parameters to procedures.

The syntax for `CREATE VARIABLE` is:

```
CREATE VARIABLE variable-name data-type
```

When a variable is created, it is initially set to NULL. You can set the value of connection-level variables in the same way as local variables, using the `SET` statement or using a `SELECT` statement with an `INTO` clause.

Connection-level variables exist until the connection is terminated, or until you explicitly drop the variable using the `DROP VARIABLE` statement. The following statement drops the variable `con_var`:

```
DROP VARIABLE con_var
```

Example

- The following batch of SQL statements illustrates the use of connection-level variables.

```
CREATE VARIABLE con_var INT;
SET con_var = 10;
MESSAGE 'con_var = ', con_var;
```

Running this batch from ISQL displays this message on the server window:

```
con_var = 10
```

Compatibility

Adaptive Server Enterprise does not support connection-level variables.

Global variables

Global variables have values set by Sybase IQ. For example, the global variable `@@version` has a value that is the current version number of the database server.

Global variables are distinguished from local and connection-level variables by two `@` signs preceding their names. For example, `@@error` is a global variable. Users cannot create global variables, and cannot update the value of global variables directly.
Some global variables, such as `@@spid`, hold connection-specific information and therefore have connection-specific values. Other variables, such as `@@connections`, have values that are common to all connections.

The special constants such as CURRENT DATE, CURRENT TIME, USER, SQLSTATE, and so on are similar to global variables.

The following statement retrieves the value of the version global variable:

```sql
SELECT @@version
```

In procedures, global variables can be selected into a variable list. The following procedure returns the server version number in the `ver` parameter.

```sql
CREATE PROCEDURE VersionProc ( OUT ver VARCHAR ( 100 ) )
BEGIN
    SELECT @@version
    INTO ver;
END
```

In Embedded SQL, global variables can be selected into a host variable list.

Table 2-5 lists the global variables available in Sybase IQ.
### Table 2-5: Sybase IQ global variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@error</td>
<td>Commonly used to check the error status (succeeded or failed) of the most recently executed statement. Contains 0 if the previous transaction succeeded; otherwise, contains the last error number generated by the system. A statement such as if @@error != 0 return causes an exit if an error occurs. Every SQL statement resets @@error, so the status check must immediately follow the statement whose success is in question.</td>
</tr>
<tr>
<td>@@fetch_status</td>
<td>Contains status information resulting from the last fetch statement. @@fetch_status may contain the following values:&lt;br&gt;  • 0 The fetch statement completed successfully.&lt;br&gt;  • -1 The fetch statement resulted in an error.&lt;br&gt;  • -2 There is no more data in the result set.&lt;br&gt;This feature is the same as @@sqlstatus, except that it returns different values. It is for Microsoft SQL Server compatibility.</td>
</tr>
<tr>
<td>@@identity</td>
<td>The last value inserted into an Identity/Autoincrement column by an insert, load or update statement. @@identity is reset each time a row is inserted into a table. If a statement inserts multiple rows, @@identity reflects the Identity/Autoincrement value for the last row inserted. If the affected table does not contain an Identity/Autoincrement column, @@identity is set to 0. The value of @@identity is not affected by the failure of an insert, load, or update statement, or the rollback of the transaction that contained the failed statement. @@identity retains the last value inserted into an Identity/Autoincrement column, even if the statement that inserted that value fails to commit.</td>
</tr>
<tr>
<td>@@isolation</td>
<td>Current isolation level. @@isolation takes the value of the active level.</td>
</tr>
<tr>
<td>@@procid</td>
<td>Stored procedure ID of the currently executing procedure.</td>
</tr>
<tr>
<td>@@servername</td>
<td>Name of the current database server.</td>
</tr>
<tr>
<td>@@sqlstatus</td>
<td>Contains status information resulting from the last FETCH statement.</td>
</tr>
<tr>
<td>@@version</td>
<td>Version number of the current version of Sybase IQ.</td>
</tr>
</tbody>
</table>
Table 2-6 includes all Adaptive Server Enterprise global variables that are supported in Sybase IQ. Adaptive Server Enterprise global variables that are not supported by Sybase IQ are not included in the list. In contrast to Table 2-5, this list includes all global variables that return a value, including those for which the value is fixed at NULL, 1, -1, or 0, and might not be meaningful.
<table>
<thead>
<tr>
<th>Global variable</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@char_convert</td>
<td>Returns 0.</td>
</tr>
<tr>
<td>@@client_cname</td>
<td>In Adaptive Server Enterprise, the client’s character set name. Set to NULL if client character set has never been initialized; otherwise, contains the name of the most recently used character set. Returns NULL in Sybase IQ.</td>
</tr>
<tr>
<td>@@client_csid</td>
<td>In Adaptive Server Enterprise, the client’s character set ID. Set to -1 if client character set has never been initialized; otherwise, contains the most recently used client character set ID from syscharsets. Returns -1 in Sybase IQ.</td>
</tr>
<tr>
<td>@@connections</td>
<td>The number of logins since the server was last started.</td>
</tr>
<tr>
<td>@@cpu_busy</td>
<td>In Adaptive Server Enterprise, the amount of time, in ticks, that the CPU has spent performing Adaptive Server Enterprise work since the last time Adaptive Server Enterprise was started. In Sybase IQ, returns 0.</td>
</tr>
<tr>
<td>@@error</td>
<td>Commonly used to check the error status (succeeded or failed) of the most recently executed statement. Contains 0 if the previous transaction succeeded; otherwise, contains the last error number generated by the system. A statement such as: if @@error != 0 return causes an exit if an error occurs. Every statement resets @@error, including PRINT statements or IF tests, so the status check must immediately follow the statement whose success is in question.</td>
</tr>
<tr>
<td>@@identity</td>
<td>In Adaptive Server Enterprise, the last value inserted into an IDENTITY column by an INSERT, LOAD, or SELECT INTO statement. @@identity is reset each time a row is inserted into a table. If a statement inserts multiple rows, @@identity reflects the IDENTITY value for the last row inserted. If the affected table does not contain an IDENTITY column, @@identity is set to 0. The value of @@identity is not affected by the failure of an INSERT or SELECT INTO statement, or the rollback of the transaction that contained the failed statement. @@identity retains the last value inserted into an IDENTITY column, even if the statement that inserted that value fails to commit.</td>
</tr>
<tr>
<td>@@idle</td>
<td>In Adaptive Server Enterprise, the amount of time, in ticks, that Adaptive Server Enterprise has been idle since the server was last started. In Sybase IQ, returns 0.</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@io_busy</td>
<td>In Adaptive Server Enterprise, the amount of time, in ticks, that Adaptive Server Enterprise has spent performing input and output operations since the server was last started. In Sybase IQ, returns 0.</td>
</tr>
<tr>
<td>@@isolation</td>
<td>Current isolation level of the connection. In Adaptive Server Enterprise, @@isolation takes the value of the active level.</td>
</tr>
<tr>
<td>@@langid</td>
<td>In Adaptive Server Enterprise, defines the local language ID of the language currently in use. In Sybase IQ, returns 0.</td>
</tr>
<tr>
<td>@@language</td>
<td>In Adaptive Server Enterprise, defines the name of the language currently in use. In Sybase IQ, returns “English”.</td>
</tr>
<tr>
<td>@@max_connections</td>
<td>For the network server, the maximum number of active clients (not database connections, as each client can support multiple connections). For Adaptive Server Enterprise, the maximum number of connections to the server.</td>
</tr>
<tr>
<td>@@ncharsize</td>
<td>In Adaptive Server Enterprise, average length, in bytes, of a national character. In Sybase IQ, returns 1.</td>
</tr>
<tr>
<td>@@nestlevel</td>
<td>In Adaptive Server Enterprise, nesting level of current execution (initially 0). Each time a stored procedure or trigger calls another stored procedure or trigger, the nesting level is incremented. In Sybase IQ, returns -1.</td>
</tr>
<tr>
<td>@@pack_received</td>
<td>In Adaptive Server Enterprise, number of input packets read by Adaptive Server Enterprise since the server was last started. In Sybase IQ, returns 0.</td>
</tr>
<tr>
<td>@@pack_sent</td>
<td>In Adaptive Server Enterprise, number of output packets written by Adaptive Server Enterprise since the server was last started. In Sybase IQ, returns 0.</td>
</tr>
<tr>
<td>@@packet_errors</td>
<td>In Adaptive Server Enterprise, number of errors that have occurred while Adaptive Server Enterprise was sending and receiving packets. In Sybase IQ, returns 0.</td>
</tr>
<tr>
<td>@@procid</td>
<td>Stored procedure ID of the currently executing procedure.</td>
</tr>
<tr>
<td>@@servername</td>
<td>Name of the local Adaptive Server Enterprise or Sybase IQ server.</td>
</tr>
<tr>
<td>@@spid</td>
<td>In Adaptive Server Enterprise, server process ID number of the current process. In Sybase IQ, the connection handle for the current connection. This is the same value as that displayed by the sa_conn_info procedure.</td>
</tr>
</tbody>
</table>
Comments

Use comments to attach explanatory text to SQL statements or statement blocks. The database server does not execute comments.

Several comment indicators are available in Sybase IQ:

- **-- (Double hyphen)** The database server ignores any remaining characters on the line. This is the SQL92 comment indicator.
Comments

- // (Double slash) The double slash has the same meaning as the double hyphen.

- /* … */ (Slash-asterisk) Any characters between the two comment markers are ignored. The two comment markers might be on the same or different lines. Comments indicated in this style can be nested. This style of commenting is also called C-style comments.

- % (Percent sign) The percent sign has the same meaning as the double hyphen. Sybase recommends that you do not use % as a comment indicator.

Note The double-hyphen and the slash-asterisk comment styles are compatible with Adaptive Server Enterprise.

Examples

- This example illustrates the use of double-dash comments:

  ```sql
  CREATE FUNCTION fullname (firstname CHAR(30),
                             lastname CHAR(30))
  RETURNS CHAR(61)
  -- fullname concatenates the firstname and lastname
  -- arguments with a single space between.
  BEGIN
    DECLARE name CHAR(61);
    SET name = firstname || ' ' || lastname;
    RETURN ( name );
  END
  ```

- This example illustrates the use of C-style comments:

  ```sql
  /*
   * Lists the names and employee IDs of employees who work in the sales department.
   */
  CREATE VIEW SalesEmployee AS
  SELECT emp_id, emp_lname, emp_fname
  FROM "GROUPO".Employees
  WHERE DepartmentID = 200
  ```
**NULL value**

**Function**
To specify a value that is unknown or not applicable

**Syntax**
NULL

**Usage**
Anywhere

**Permissions**
Must be connected to the database

**Side effects**
None

**Description**
The NULL value is a special value that is different from any valid value for any data type. However, the NULL value is a legal value in any data type. The NULL value is used to represent missing or inapplicable information. These are two separate and distinct cases where NULL is used:

<table>
<thead>
<tr>
<th>Situation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missing</td>
<td>The field does have a value, but that value is unknown.</td>
</tr>
<tr>
<td>inapplicable</td>
<td>The field does not apply for this particular row.</td>
</tr>
</tbody>
</table>

SQL allows columns to be created with the NOT NULL restriction. This means that those particular columns cannot contain the NULL value.

The NULL value introduces the concept of three valued logic to SQL. The NULL value compared using any comparison operator with any value including the NULL value is UNKNOWN. The only search condition that returns TRUE is the IS NULL predicate. In SQL, rows are selected only if the search condition in the WHERE clause evaluates to TRUE; rows that evaluate to UNKNOWN or FALSE are not selected.

You can also use the IS [ NOT ] truth-value clause, where truth-value is one of TRUE, FALSE or UNKNOWN, to select rows where the NULL value is involved. See “Search conditions” on page 33 for a description of this clause.

In the following examples, the column Salary contains the NULL value.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Truth value</th>
<th>Selected?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salary = NULL</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>Salary &lt;&gt; NULL</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>NOT (Salary = NULL)</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>NOT (Salary &lt;&gt; NULL)</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>Salary = 1000</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>Salary IS NULL</td>
<td>TRUE</td>
<td>YES</td>
</tr>
<tr>
<td>Salary IS NOT NULL</td>
<td>FALSE</td>
<td>NO</td>
</tr>
</tbody>
</table>
The NULL value also has an interesting property when used in numeric expressions. The result of *any* numeric expression involving the NULL value is the NULL value. This means that if the NULL value is added to a number, the result is the NULL value—not a number. If you want the NULL value to be treated as 0, you must use the `ISNULL( expression, 0 )` function. See Chapter 4, “SQL Functions.”

Many common errors in formulating SQL queries are caused by the behavior of NULL. Be careful to avoid these problem areas. See “Search conditions” on page 33 for a description of the effect of three-valued logic when combining search conditions.

### Example

The following `INSERT` statement inserts a NULL into the `date_returned` column of the `Borrowed_book` table.

```sql
INSERT
  INTO Borrowed_book
  ( date_borrowed, date_returned, book )
VALUES ( CURRENT_DATE, NULL, '1234' )
```
CHAPTER 3

SQL Data Types

About this chapter

This chapter describes the data types supported by Sybase IQ.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character data types</td>
<td>69</td>
</tr>
<tr>
<td>Numeric data types</td>
<td>73</td>
</tr>
<tr>
<td>Binary data types</td>
<td>77</td>
</tr>
<tr>
<td>Bit data type</td>
<td>82</td>
</tr>
<tr>
<td>Date and time data types</td>
<td>82</td>
</tr>
<tr>
<td>Sending dates and times to the database</td>
<td>84</td>
</tr>
<tr>
<td>Retrieving dates and times from the database</td>
<td>85</td>
</tr>
<tr>
<td>Comparing dates and times</td>
<td>85</td>
</tr>
<tr>
<td>Using unambiguous dates and times</td>
<td>86</td>
</tr>
<tr>
<td>Domains</td>
<td>87</td>
</tr>
<tr>
<td>Data type conversions</td>
<td>89</td>
</tr>
<tr>
<td>Sybase IQ binary load format</td>
<td>94</td>
</tr>
</tbody>
</table>

Character data types

Description

For storing strings of letters, numbers and symbols.

Syntax

- `CHAR [ ( max-length ) ]`
- `CHARACTER [ ( max-length ) ]`
- `CHARACTER VARYING [ ( max-length ) ]`
- `VARCHAR [ ( max-length ) ]`
- `UNIQUEIDENTIFIERSTR`

Usage

- `CHAR` Character data of maximum length `max-length` bytes. If `max-length` is omitted, the default is 1. The maximum size allowed is 32KB – 1. See Notes for restrictions on CHAR data greater than 255 bytes.

See the notes below on character data representation in the database, and on storage of long strings.
Character data types

All CHAR values are blank padded up to max-length, regardless of whether the BLANK PADDING option is specified. When multibyte character strings are held as a CHAR type, the maximum length is still in bytes, not characters.

**CHARACTER**  Same as CHAR.

**CHARACTER VARYING**  Same as VARCHAR.

**LONG VARCHAR**  Arbitrary length character data. The maximum size is limited by the maximum size of the database file (currently 2 gigabytes).

**TEXT**  This is a user-defined data type. It is implemented as a LONG VARCHAR allowing NULL.

**VARCHAR**  Same as CHAR, except that no blank padding is added to the storage of these strings, and VARCHAR strings can have a maximum length of (32KB – 1). See Notes for restrictions on VARCHAR data greater than 255 bytes.

**UNIQUEIDENTIFIERSTR**  Domain implemented as CHAR(36). This data type is used for remote data access, when mapping Microsoft SQL Server uniqueidentifier columns.

Notes

As a separately licensed option, Sybase IQ supports Character Large Object (CLOB) data with a length ranging from zero (0) to 512TB (terabytes) for an IQ page size of 128KB or 2PB (petabytes) for an IQ page size of 512KB. The maximum length is equal to 4GB multiplied by the database page size. See Large Objects Management in Sybase IQ.

**Storage sizes**

Table 3-1 lists the storage size of character data.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Column definition</th>
<th>Input data</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER, CHAR</td>
<td>width of (32K – 1) bytes</td>
<td>(32K – 1) bytes</td>
<td>(32K – 1) bytes</td>
</tr>
<tr>
<td>VARCHAR, CHARACTER VARYING</td>
<td>width of (32K – 1) bytes</td>
<td>(32K – 1) bytes</td>
<td>(32K – 1) bytes</td>
</tr>
</tbody>
</table>

**Character sets and code pages**

Character data is placed in the database using the exact binary representation that is passed from the application. This usually means that character data is stored in the database with the binary representation of the character set used by your system. You can find documentation about character sets in the documentation for your operating system.
On Windows, code pages are the same for the first 128 characters. If you use special characters from the top half of the code page (accented international language characters), you must be careful with your databases. In particular, if you copy the database to a different machine using a different code page, those special characters are retrieved from the database using the original code page representation. With the new code page, they appear on the screen to be the wrong characters.

This problem also appears if you have two clients using the same multiuser server, but running with different code pages. Data inserted or updated by one client might appear incorrect to another.

This problem also shows up if a database is used across platforms. PowerBuilder and many other Windows applications insert data into the database in the standard ANSI character set. If non-Windows applications attempt to use this data, they do not properly display or update the extended characters.

This problem is quite complex. If any of your applications use the extended characters in the upper half of the code page, make sure that all clients and all machines using the database use the same or a compatible code page.

**Indexes**

All index types, except `DATE`, `TIME`, and `DTTM`, are supported for `CHAR` data and `VARCHAR` data less than or equal to 255 bytes in length.

**VARCHAR data and trailing blanks**

Data inserted using `INSERT`, `UPDATE`, or `LOAD TABLE` can be:

- Enclosed in quotes
- Not enclosed in quotes
- Binary

For a column of data type `VARCHAR`, trailing blanks within the data being inserted are handled as follows:

- For data enclosed in quotes, trailing blanks are never trimmed.
- For data not enclosed in quotes:
  - Trailing blanks always trimmed on insert and update.
  - For a `LOAD` statement, you can use the `STRIP RTRIM/OFF LOAD` option to specify whether to have the trailing blanks trimmed. The `STRIP RTRIM/OFF` option applies only to variable-length non-binary data. For example, assume the following schema:

```sql
CREATE TABLE t( c1 VARCHAR(3) );
LOAD TABLE t( c1 ',' ) ......... STRIP RTRIM  // trailing blanks trimmed
LOAD TABLE t( c1 ',' ) ......... STRIP OFF    // trailing blanks not trimmed
```
Character data types

LOAD TABLE t( c1 ASCII(3) ) ... STRIP RTRIM // trailing blanks not trimmed
LOAD TABLE t( c1 ASCII(3) ) ... STRIP OFF // trailing blanks trimmed
LOAD TABLE t( c1 BINARY ) ..... STRIP RTRIM // trailing blanks trimmed
LOAD TABLE t( c1 BINARY ) ..... STRIP OFF // trailing blanks trimmed

- For binary data, trailing blanks are always trimmed.

When you write your applications, do not depend on the existence of trailing blanks in VARCHAR columns. If an application relies on trailing blanks, use a CHAR column instead of a VARCHAR column.

Restriction on CHAR and VARCHAR data over 255 bytes

Only the default index, WD, and CMP index types are supported for CHAR and VARCHAR columns over 255 bytes. You cannot create an LF, HG, HNG, DATE, TIME, or DTTM index for these columns.

Compatibility

- The CHARACTER (n) alternative for CHAR is not supported in Adaptive Server Enterprise.
- Sybase IQ does not support the NCHAR, NVARCHAR, UNICHAR, and UNIVARCHAR data types provided by Adaptive Server Enterprise. Sybase IQ supports Unicode in the CHAR and VARCHAR data types.
- Sybase IQ supports a longer LONG VARCHAR data type than SQL Anywhere. See Large Objects Management in Sybase IQ.
- For compatibility between Sybase IQ and Adaptive Server Enterprise, always specify a length for character data types.

Long strings

SQL Anywhere treats CHAR, VARCHAR, and LONG VARCHAR columns all as the same type. Values up to 254 characters are stored as short strings, with a preceding length byte. Any values that are longer than 255 bytes are considered long strings. Characters after the 255th are stored separate from the row containing the long string value.

There are several functions (see SQL Functions) that will ignore the part of any string past the 255th character. They are soundex, similar, and all of the date functions. Also, any arithmetic involving the conversion of a long string to a number will work on only the first 255 characters. It would be extremely unusual to run into one of these limitations.

All other functions and all other operators work with the full length of long strings.
## Numeric data types

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| For storing numerical data. | [ UNSIGNED ] BIGINT  
| | [ UNSIGNED ] { INT | INTEGER }  
| | SMALLINT  
| | TINYINT  
| | DECIMAL [ ( precision [ , scale ] ) ]  
| | NUMERIC [ ( precision [ , scale ] ) ]  
| | DOUBLE  
| | FLOAT [ ( precision ) ]  
| | REAL |

### Usage

**BIGINT** A signed 64-bit integer, requiring 8 bytes of storage.

You can specify integers as UNSIGNED. By default the data type is signed. Its range is between -9223372036854775808 and 9223372036854775807 (signed) or from 0 to 18446744073709551615 (unsigned).

**INT or INTEGER** A signed 32-bit integer with a range of values between -2147483648 and 2147483647 requiring 4 bytes of storage.

The INTEGER data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

You can specify integers as UNSIGNED; by default the data type is signed. The range of values for an unsigned integer is between 0 and 4294967295.

**SMALLINT** A signed 16-bit integer with a range between -32768 and 32767, requiring 2 bytes of storage.

The SMALLINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

**TINYINT** An unsigned 8-bit integer with a range between 0 and 255, requiring 1 byte of storage.

The TINYINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.
**DECIMAL**  A signed decimal number with *precision* total digits and with *scale* of the digits after the decimal point. The precision can equal 1 to 126, and the scale can equal 0 up to precision value. The defaults are scale = 38 and precision = 126. Results are calculated based on the actual data type of the column to ensure accuracy, but you can set the maximum scale of the result returned to the application. For more information, see the “MAX_CLIENT_NUMERIC_SCALE option” on page 420 and the SET OPTION statement in Reference: Statements and Options.

Table 3-2 lists the storage required for a decimal number.

<table>
<thead>
<tr>
<th>Precision</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 4</td>
<td>2 bytes</td>
</tr>
<tr>
<td>5 to 9</td>
<td>4 bytes</td>
</tr>
<tr>
<td>10 to 18</td>
<td>8 bytes</td>
</tr>
<tr>
<td>19 to 126</td>
<td>See below</td>
</tr>
</tbody>
</table>

The storage requirement in bytes for a decimal value with a precision greater than 18 can be calculated using the following formula:

\[ 4 + 2 \times (\text{int}((\text{prec} - \text{scale}) + 3) / 4) + \text{int}((\text{scale} + 3) / 4) + 1) \]

where *int* takes the integer portion of its argument. The storage used by a column is based upon the precision and scale of the column. Each cell in the column has enough space to hold the largest value of that precision and scale. For example:

- **NUMERIC(18,4)** takes 8 bytes per cell
- **NUMERIC(19,4)** takes 16 bytes per cell

The **DECIMAL** data type is an exact numeric data type; its accuracy is preserved to the least significant digit after arithmetic operations. Its maximum absolute value is the number of nines defined by \([\text{precision} - \text{scale}]\), followed by the decimal point, and then followed by the number of nines defined by *scale*. The minimum absolute nonzero value is the decimal point, followed by the number of zeros defined by \([\text{scale} - 1]\), then followed by a single one. For example:

- **NUMERIC(3,2)** Max positive = 9.99 Min non-zero = 0.01 Max negative = -9.99

If neither precision nor scale is specified for the explicit conversion of NULL to **NUMERIC**, the default is **NUMERIC(1,0)**. For example,

```sql
SELECT CAST( NULL AS NUMERIC ) A,
       CAST( NULL AS NUMERIC(15,2) ) B
```
is described as:

A NUMERIC(1, 0)
B NUMERIC(15, 2)

**NUMERIC**  Same as DECIMAL.

**DOUBLE**  A signed double-precision floating-point number stored in 8 bytes. The range of absolute, nonzero values is between 2.2250738585072014e-308 and 1.797693134862315708e+308. Values held as DOUBLE are accurate to 15 significant digits, but might be subject to rounding errors beyond the fifteenth digit.

The DOUBLE data type is an approximate numeric data type; it is subject to rounding errors after arithmetic operations.

**FLOAT**  If precision is not supplied, the FLOAT data type is the same as the REAL data type. If precision supplied, then the FLOAT data type is the same as the REAL or DOUBLE data type, depending on the value of the precision. The cutoff between REAL and DOUBLE is platform-dependent, and it is the number of bits used in the mantissa of single-precision floating point number on the platform.

When a column is created using the FLOAT data type, columns on all platforms are guaranteed to hold the values to at least the specified minimum precision. In contrast, REAL and DOUBLE do not guarantee a platform-independent minimum precision.

The FLOAT data type is an approximate numeric data type; it is subject to rounding errors after arithmetic operations.

**REAL**  A signed single-precision floating-point number stored in 4 bytes. The range of absolute, nonzero values is 1.175494351e-38 to 3.402823466e+38. Values held as REAL are accurate to 6 significant digits, but might be subject to rounding errors beyond the sixth digit.

The REAL data type is an approximate numeric data type; it is subject to rounding errors after arithmetic operations.

**Notes**

- The INTEGER, NUMERIC, and DECIMAL data types are sometimes called exact numeric data types, in contrast to the approximate numeric data types FLOAT, DOUBLE, and REAL. Only exact numeric data is guaranteed to be accurate to the least significant digit specified after arithmetic operations.
**Numeric data types**

- Do not fetch TINYINT columns into Embedded SQL variables defined as CHAR or UNSIGNED CHAR, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program.

**Indexes**

- The CMP and HNG index types do not support the FLOAT, DOUBLE, and REAL data types, and the HG index type is not recommended.

- The WD, DATE, TIME, and DTTM index types do not support the numeric data types.

**Compatibility**

- In embedded SQL, fetch TINYINT columns into 2-byte or 4-byte integer columns. Also, to send a TINYINT value to a database, the C variable should be an integer.

- Adaptive Server Enterprise 12.5.x versions do not support unsigned integers. You can map Sybase IQ unsigned integers to Adaptive Server Enterprise signed integers or numeric data, and the data are converted implicitly.
  - Map IQ UNSIGNED SMALLINT data to ASE INT
  - If you have negative values, map IQ UNSIGNED BIGINT to ASE NUMERIC (precision, scale)

  To avoid performance issues for cross-database joins on UNSIGNED BIGINT columns, the best approach is to cast to a (signed) BIGINT on the Sybase IQ side.

- You should avoid default precision and scale settings for NUMERIC and DECIMAL data types, as these differ by product:

<table>
<thead>
<tr>
<th>Database</th>
<th>Default precision</th>
<th>Default scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sybase IQ</td>
<td>126</td>
<td>38</td>
</tr>
<tr>
<td>Adaptive Server Enterprise</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>SQL Anywhere</td>
<td>30</td>
<td>6</td>
</tr>
</tbody>
</table>

- The FLOAT (p) data type is a synonym for REAL or DOUBLE, depending on the value of p. For Adaptive Server Enterprise, REAL is used for p less than or equal to 15, and DOUBLE for p greater than 15. For Sybase IQ, the cutoff is platform-dependent, but on all platforms, the cutoff value is greater than 22.
• Sybase IQ includes two user-defined data types, MONEY and SMALLMONEY, which are implemented as NUMERIC(19,4) and NUMERIC(10,4) respectively. They are provided primarily for compatibility with Adaptive Server Enterprise.

**Binary data types**

**Description**
For storing raw binary data, such as pictures, in a hexadecimal-like notation, up to a length of (32K – 1) bytes. The UNIQUEIDENTIFIER data type is used for storage of UUID (also known as GUID) values.

**Syntax**

- **BINARY** [(length)]
- **VARBINARY** [(max-length)]
- **UNIQUEIDENTIFIER**

**Usage**
Binary data begins with the characters “0x” or “0X” and can include any combination of digits and the uppercase and lowercase letters A through F. You can specify the column length in bytes, or use the default length of 1 byte. Each byte stores 2 hexadecimal digits. Even though the default length is 1 byte, Sybase recommends that you always specify an even number of characters for BINARY and VARBINARY column length. If you enter a value longer than the specified column length, Sybase IQ truncates the entry to the specified length without warning or error.

**BINARY**
Binary data of length *length* bytes. If *length* is omitted, the default is 1 byte. The maximum size allowed is 255 bytes. Use the fixed-length binary type BINARY for data in which all entries are expected to be approximately equal in length. Because entries in BINARY columns are zero-padded to the column length *length*, they might require more storage space than entries in VARBINARY columns.

**VARBINARY**
Binary data up to a length of *max-length* bytes. If *max-length* is omitted, the default is 1 byte. The maximum size allowed is (32K – 1) bytes. Use the variable-length binary type VARBINARY for data that is expected to vary greatly in length.

**Notes**
As a separately licensed option, Sybase IQ supports binary large object (BLOB) data with a length ranging from zero (0) to 512TB (terabytes) for an IQ page size of 128KB or 2PB (petabytes) for an IQ page size of 512KB. The maximum length is equal to 4GB multiplied by the database page size. See *Large Objects Management in Sybase IQ*.
**Binary data types**

For information on LONG BINARY and IMAGE data types, see “Binary data types” on page 603.

All BINARY columns are padded with zeros to the full width of the column. Trailing zeros are truncated in all VARBINARY columns.

The following example creates a table with all four variations of BINARY and VARBINARY data types defined with NULL and NOT NULL. The same data is inserted in all four columns and is padded or truncated according to the data type of the column.

```sql
CREATE TABLE zeros (bnot BINARY(5) NOT NULL,
    bnull BINARY(5) NULL,
    vbnot VARBINARY(5) NOT NULL,
    vbnull VARBINARY(5) NULL);
INSERT zeros VALUES (0x12345000, 0x12345000, 0x12345000, 0x12345000);
INSERT zeros VALUES (0x123, 0x123, 0x123, 0x123);
INSERT zeros VALUES (0x0, 0x0, 0x0, 0x0);
INSERT zeros VALUES ('002710000000ae1b', '002710000000ae1b', '002710000000ae1b', '002710000000ae1b');
SELECT * FROM zeros;
```

Because each byte of storage holds 2 hexadecimal digits, Sybase IQ expects binary entries to consist of the characters “0x” followed by an even number of digits. When the “0x” is followed by an odd number of digits, Sybase IQ assumes that you omitted the leading 0 and adds it for you.

Input values “0x00” and “0x0” are stored as “0x00” in variable-length binary columns (VARBINARY). In fixed-length binary columns (BINARY), the value is padded with zeros to the full length of the field:

```sql
INSERT zeros VALUES (0x0, 0x0, 0x0, 0x0);
SELECT * FROM zeros
```

<table>
<thead>
<tr>
<th>bnot</th>
<th>bnull</th>
<th>vbnot</th>
<th>vbnull</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234500000</td>
<td>0x1234500000</td>
<td>0x12345000</td>
<td>0x12345000</td>
</tr>
<tr>
<td>0x0123000000</td>
<td>0x0123000000</td>
<td>0x0123</td>
<td>0x0123</td>
</tr>
<tr>
<td>0x0000000000</td>
<td>0x0000000000</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>0x3030323731</td>
<td>0x3030323731</td>
<td>0x3030323731</td>
<td>0x3030323731</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bnot</th>
<th>bnull</th>
<th>vbnot</th>
<th>vbnull</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000000</td>
<td>0x0000000000</td>
<td>0x00</td>
<td>0x00</td>
</tr>
</tbody>
</table>
If the input value does not include “0x”, Sybase IQ assumes that the value is an ASCII value and converts it. For example:

```sql
CREATE TABLE sample (col_bin BINARY(8));
INSERT sample VALUES ('002710000000ae1b');
SELECT * FROM sample;
```

**col_bin**

0x3030323731303030

**Note** In the above example, ensure you set the string_rtruncation option to “off”.

When you select a BINARY value, you must specify the value with the padded zeros or use the CAST function. For example:

```sql
SELECT * FROM zeros WHERE bnot = 0x0123000000;
```

or:

```sql
SELECT * FROM zeros WHERE bnot = CAST(0x0123 as binary(5));
```

Any ASCII data loaded from a flat file into a binary type column (BINARY or VARBINARY) is stored as nibbles. For example, if 0x1234 or 1234 is read from a flat file into a binary column, Sybase IQ stores the value as hexadecimal 1234. Sybase IQ ignores the “0x” prefix. If the input data contains any characters out of the range 0 – 9, a – f, and A – F, the data is rejected.

Storage size

Table 3-3 lists the storage size of binary data.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Column definition</th>
<th>Input data</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARBINARY</td>
<td>width of (32K – 1) bytes</td>
<td>(32K – 1) bytes binary</td>
<td>(32K – 1) bytes</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>width of (32K – 1) bytes</td>
<td>(64K – 2) bytes ASCII</td>
<td>(32K – 1) bytes</td>
</tr>
<tr>
<td>BINARY</td>
<td>width of 255 bytes</td>
<td>255 bytes binary</td>
<td>255 bytes</td>
</tr>
<tr>
<td>BINARY</td>
<td>width of 255 bytes</td>
<td>510 bytes ASCII</td>
<td>255 bytes</td>
</tr>
</tbody>
</table>

**Platform dependence** The exact form in which you enter a particular value depends on the platform you are using. Therefore, calculations involving binary data might produce different results on different machines.
Binary data types

For platform-independent conversions between hexadecimal strings and integers, use the INTTOHEX and HEXTOINT functions rather than the platform-specific CONVERT function. For details, see the section “Data type conversion functions” on page 108.

String operators

The concatenation string operators || and + both support binary type data. Explicit conversion of binary operands to character data types is not necessary with the || operator. Explicit and implicit data conversion produce different results, however.

Restrictions on BINARY and VARBINARY data

The following restrictions apply to columns containing BINARY and VARBINARY data:

- You cannot use the aggregate functions SUM, AVG, STDDEV, or VARIANCE with the binary data types. The aggregate functions MIN, MAX, and COUNT do support the binary data types BINARY and VARBINARY.

- HNG, WD, DATE, TIME, and DTTM indexes do not support BINARY or VARBINARY data.

- Only the default index and CMP index types are supported for VARBINARY data greater than 255 bytes in length.

- Bit operations are supported on BINARY and VARBINARY data that is 8 bytes or less in length.

Compatibility

The treatment of trailing zeros in binary data differs between Sybase IQ, SQL Anywhere, and Adaptive Server Enterprise:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Sybase IQ</th>
<th>SQL Anywhere</th>
<th>ASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY NOT NULL</td>
<td>Padded</td>
<td>Not padded</td>
<td>Padded</td>
</tr>
<tr>
<td>BINARY NULL</td>
<td>Padded</td>
<td>Not padded</td>
<td>Not padded</td>
</tr>
<tr>
<td>VARBINARY NOT NULL</td>
<td>Truncated, not padded</td>
<td>Truncated, not padded</td>
<td>Truncated, not padded</td>
</tr>
<tr>
<td>VARBINARY NULL</td>
<td>Truncated, not padded</td>
<td>Truncated, not padded</td>
<td>Truncated, not padded</td>
</tr>
</tbody>
</table>

Adaptive Server Enterprise, SQL Anywhere, and Sybase IQ all support the STRING_RTRUNCATION database option, which affects error message reporting when an INSERT or UPDATE string is truncated. For Transact-SQL compatible string comparisons, set the STRING_RTRUNCATION option to the same value in both databases.
You can also set the STRING_RTRUNCATION option ON when loading data into a table, to alert you that the data is too large to load into the field. The default value is ON.

Bit operations on binary type data are not supported by Adaptive Server Enterprise. SQL Anywhere only supports bit operations against the first four bytes of binary type data. Sybase IQ supports bit operations against the first eight bytes of binary type data.

**UNIQUEIDENTIFIER** Used for storage of UUID (also known as GUID) values. The UNIQUEIDENTIFIER data type is often used for a primary key or other unique column to hold UUID (Universally Unique Identifier) values that can be used to uniquely identify rows. The NEWID function generates UUID values in such a way that a value produced on one computer does not match a UUID produced on another computer. UNIQUEIDENTIFIER values generated using NEWID can therefore be used as keys in a synchronization environment.

For example, the following statement updates the table mytab and sets the value of the column uid_col to a unique identifier generated by the NEWID function, if the current value of the column is NULL.

```sql
UPDATE mytab
SET uid_col = NEWID()
WHERE uid_col IS NULL
```

If you execute the following statement,

```sql
SELECT NEWID()
```

the unique identifier is returned as a BINARY(16). For example, the value might be 0x3749fe09cf446e399913be6434f1f08. You can convert this string into a readable format using the UUIDTOSTR() function.

UUID values are also referred to as GUIDs (Globally Unique Identifier).

The STRTOUUID and UUIDTOSTR functions are used to convert values between UNIQUEIDENTIFIER and string representations.

UNIQUEIDENTIFIER values are stored and returned as BINARY(16).

Because UNIQUEIDENTIFIER values are large, using UNSIGNED BIGINT or UNSIGNED INT identity columns instead of UNIQUEIDENTIFIER is more efficient, if you do not need cross database unique identifiers.

Standards and compatibility for UNIQUEIDENTIFIER

- **SQL92** Vendor extension.
- **Sybase** Supported by SQL Anywhere. Not supported by Adaptive Server Enterprise.
Bit data type

- **Backwards compatibility** In databases created before Sybase IQ version 12.7, the STRTOUUID, UUIDTOSTR, and NEWID functions were supported through CIS functional compensation. In versions 15.1 and later, the STRTOUUID, UUIDTOSTR, and NEWID functions are native Sybase IQ functions.

See also
For more information related to UNIQUEIDENTIFIER:
- “NEWID function [Miscellaneous]” on page 208
- “UUIDTOSTR function [String]” on page 331
- “STRTOUUID function [String]” on page 263

Bit data type

Description
For storing Boolean values.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Values</th>
<th>Supported by</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>0 or 1</td>
<td>Sybase IQ and Enterprise</td>
</tr>
</tbody>
</table>

Usage
BIT stores only the values 0 or 1. Inserting any nonzero value into a BIT column stores a 1 in the column. Inserting any zero value into a BIT column stores a 0.

Only the default index type is supported for BIT data.

Compatibility
Adaptive Server Enterprise BIT datatypes only allow 0 or 1 values.

Date and time data types

Description
For storing dates and times.

Syntax
- DATE
- DATETIME
- SMALLDATETIME
- TIME
- TIMESTAMP
Usage

**DATE**  A calendar date, such as a year, month and day. The year can be from 0001 to 9999. The day must be a nonzero value, so that the minimum date is 0001-01-01. A DATE value requires 4 bytes of storage.

**DATETIME**  A domain, implemented as TIMESTAMP. DATETIME is provided primarily for compatibility with Adaptive Server Enterprise. For an exception, see “Compatibility of string to datetime conversions” on page 90.

**SMALLDATETIME**  A domain, implemented as TIMESTAMP. SMALLDATETIME is provided primarily for compatibility with Adaptive Server Enterprise. For an exception, see “Compatibility of string to datetime conversions” on page 90.

**TIME**  Time of day, containing hour, minute, second, and fraction of a second. The fraction is stored to 6 decimal places. A TIME value requires 8 bytes of storage. (ODBC standards restrict TIME data type to an accuracy of seconds. For this reason, do not use TIME data types in WHERE clause comparisons that rely on a higher accuracy than seconds.)

**TIMESTAMP**  Point in time, containing year, month, day, hour, minute, second, and fraction of a second. The day must be a nonzero value. A TIMESTAMP value requires 8 bytes of storage.

The valid range of the TIMESTAMP data type is from 0001-01-01 00:00:00.000000 to 9999-12-31 23:59:59.999999. The display of TIMESTAMP data outside the range of 1600-02-28 23:59:59 to 7911-01-01 00:00:00 might be incomplete, but the complete datetime value is stored in the database; you can see the complete value by first converting it to a character string. You can use the CAST() function to do this, as in the following example, which first creates a table with DATETIME and TIMESTAMP columns, then inserts values where the date is greater 7911-01-01.

```sql
create table mydates (id int, descript char(20),
                      datetime_null datetime, timestamp_null timestamp);
insert into mydates values (1, 'example', '7911-12-30 23:59:59', '7911-12-30 06:03:44');
commit;

When you select without using CAST, hours and minutes are set to 00:00:

```sql
select * from mydates;
```

1, 'example', '7911-12-30 00:00:59.00', '7911-12-30 00:00:44.00'

When you select using cast, you see the complete timestamp:

```sql
select id, descript, cast(datetime_null as char(21)),
cast(timestamp_null as char(21)) from mydates;
```
Sending dates and times to the database

Description

You can send dates and times to the database in one of the following ways:

- Using any interface, as a string
- Using ODBC, as a TIMESTAMP structure
- Using Embedded SQL, as a SQLDATETIME structure

When you send a time to the database as a string (for the TIME data type) or as part of a string (for TIMESTAMP or DATE data types), hours, minutes, and seconds must be separated by colons in the format $hh:mm:ss:sss$, but can appear anywhere in the string. As an option, a period can separate the seconds from fractions of a second, as in $hh:mm:ss.sss$. The following are valid and unambiguous strings for specifying times:

- 21:35  -- 24 hour clock if no am or pm specified
- 10:00pm  -- pm specified, so interpreted as 12 hour clock
- 10:00  -- 10:00am in the absence of pm
- 10:23:32.234  -- seconds and fractions of a second included

When you send a date to the database as a string, conversion to a date is automatic. You can supply the string in one of two ways:

- As a string of format $yyyy/mm/dd$ or $yyyy-mm-dd$, which is interpreted unambiguously by the database
- As a string interpreted according to the DATE_ORDER database option
Date format strings cannot contain any multibyte characters. Only single-byte characters are allowed in a date/time/datetime format string, even when the collation order of the database is a multibyte collation order like 932JPN.

Retrieving dates and times from the database

Description
You can retrieve dates and times from the database in one of the following ways:
- Using any interface, as a string
- Using ODBC, as a `TIMESTAMP` structure
- Using embedded SQL, as a `SQLDATETIME` structure

When a date or time is retrieved as a string, it is retrieved in the format specified by the database options `DATE_FORMAT`, `TIME_FORMAT` and `TIMESTAMP_FORMAT`. For descriptions of these options, see SET OPTION statement in Reference: Statements and Options.

For information on functions dealing with dates and times, see “Date and time data types” on page 82. The following operators are allowed on dates:

- `timestamp + integer`  Add the specified number of days to a date or timestamp.
- `timestamp - integer`  Subtract the specified number of days from a date or timestamp.
- `date - date`  Compute the number of days between two dates or timestamps.
- `date + time`  Create a timestamp combining the given date and time.

Comparing dates and times

Description
To compare a date to a string as a string, use the `DATEFORMAT` function or `CAST` function to convert the date to a string before comparing. For example:

```
DATEFORMAT(invoice_date,'yyyy/mm/dd') = '1992/05/23'
```

You can use any allowable date format for the `DATEFORMAT` string expression.
Using unambiguous dates and times

Date format strings must not contain any multibyte characters. Only single-byte characters are allowed in a date/time/datetime format string, even when the collation order of the database is a multibyte collation order like 932JPN. If ‘?’ represents a multibyte character, the following query fails:

```sql
SELECT DATEFORMAT (StartDate, 'yy?') FROM Employees;
```

Instead, move the multibyte character outside of the date format string using the concatenation operator:

```sql
SELECT DATEFORMAT (StartDate, 'yy') + '?' FROM Employees;
```

Using unambiguous dates and times

**Description**

Dates in the format `yyyy/mm/dd` or `yyyy-mm-dd` are always recognized as dates regardless of the `DATE_ORDER` setting. You can use other characters as separators; for example, a question mark, a space character, or a comma. You should use this format in any context where different users might be employing different `DATE_ORDER` settings. For example, in stored procedures, use of the unambiguous date format prevents misinterpretation of dates according to the user's `DATE_ORDER` setting.

A string of the form `hh:mm:ss.sss` is also interpreted unambiguously as a time.

For combinations of dates and times, any unambiguous date and any unambiguous time yield an unambiguous date-time value. Also, the following form is an unambiguous date-time value:

```
YYYY-MM-DD HH.MM.SS.SSSSSS
```

You can use periods in the time only in combination with a date.

In other contexts, you can use a more flexible date format. Sybase IQ can interpret a wide range of strings as formats. The interpretation depends on the setting of the database option `DATE_ORDER`. The `DATE_ORDER` database option can have the value ‘MDY’, ‘YMD’, or ‘DMY’. See SET OPTION statement in Reference: Statements and Options. For example, to set the `DATE_ORDER` option to ‘DMY’ enter:

```sql
SET OPTION DATE_ORDER = 'DMY' ;
```

The default `DATE_ORDER` setting is ‘YMD’. The ODBC driver sets the `DATE_ORDER` option to ‘YMD’ whenever a connection is made. Use the SET OPTION statement to change the value.
The database option `DATE_ORDER` determines whether the string 10/11/12 is interpreted by the database as Oct 11 1912, Nov 12 1910, or Nov 10 1912. The year, month, and day of a date string should be separated by some character (for example “/”, “-”, or space) and appear in the order specified by the `DATE_ORDER` option.

You can supply the year as either 2 or 4 digits. The value of the option `NEAREST_CENTURY` affects the interpretation of 2-digit years: 2000 is added to values less than `NEAREST_CENTURY`, and 1900 is added to all other values. The default value of this option is 50. Thus, by default, 50 is interpreted as 1950, and 49 is interpreted as 2049. See the “NEAREST_CENTURY option [TSQL]” in Reference: Statements and Options.

The month can be the name or number of the month. The hours and minutes are separated by a colon, but can appear anywhere in the string.

Sybase recommends that you always specify the year using the 4-digit format.

With an appropriate setting of `DATE_ORDER`, the following strings are all valid dates:

- 99-05-23 21:35
- 99/5/23
- 1999/05/23
- May 23 1999
- 23-May-1999
- Tuesday May 23, 1999 10:00pm

If a string contains only a partial date specification, default values are used to fill out the date. The following defaults are used:

- **year** 1900
- **month** No default
- **day** 1 (useful for month fields; for example, ‘May 1999’ is the date ‘1999-05-01 00:00’)
- **hour, minute, second, fraction** 0

## Domains

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domains are aliases for built-in data types, including precision and scale values where applicable.</td>
</tr>
</tbody>
</table>

Reference: Building Blocks, Tables, and Procedures
Domains, also called user-defined data types, allow columns throughout a database to be defined automatically on the same data type, with the same NULL or NOT NULL condition. This encourages consistency throughout the database. Domain names are case-insensitive. Sybase IQ returns an error if you attempt to create a domain with the same name as an existing domain except for case.

Simple domains

You create domains using the `CREATE DOMAIN` statement. See “CREATE DOMAIN statement” in Reference: Statements and Options.

The following statement creates a data type named `street_address`, which is a 35-character string:

```sql
CREATE DOMAIN street_address CHAR( 35 )
```

You can use `CREATE DATATYPE` as an alternative to `CREATE DOMAIN`, but this is not recommended, as `CREATE DOMAIN` is the syntax used in the draft SQL/3 standard.

Resource authority is required to create data types. Once a data type is created, the user ID that executed the `CREATE DOMAIN` statement is the owner of that data type. Any user can use the data type, and unlike other database objects, the owner name is never used to prefix the data type name.

The `street_address` data type may be used in exactly the same way as any other data type when defining columns. For example, the following table with two columns has the second column as a `street_address` column:

```sql
CREATE TABLE twocol (id INT,
                      street street_address)
```

Owners or DBAs can drop domains by issuing a `COMMIT` and then using the `DROP DOMAIN` statement:

```sql
DROP DOMAIN street_address
```

You can carry out this statement only if no tables in the database are using data type.

Many of the attributes associated with columns, such as allowing NULL values, having a DEFAULT value, and so on, can be built into a user-defined data type. Any column that is defined on the data type automatically inherits the NULL setting, CHECK condition, and DEFAULT values. This allows uniformity to be built into columns with a similar meaning throughout a database.

For example, many primary key columns in the sample database are integer columns holding ID numbers. The following statement creates a data type that may be useful for such columns:
CREATE DOMAIN id INT
    NOT NULL
    DEFAULT AUTOINCREMENT
    CHECK( @col > 0 )

Any column created using the data type id is not allowed to hold NULLs,
defaults to an autoincremented value, and must hold a positive number. Any
identifier could be used instead of col in the @col variable.

The attributes of the data type can be overridden if needed by explicitly
providing attributes for the column. A column created on data type id with
NULL values explicitly allowed does allow NULLs, regardless of the setting
in the id data type.

Compatibility

- **Named constraints and defaults**  In Sybase IQ, user-defined data types
  are created with a base data type, and optionally, a NULL or NOT NULL
  condition. Named constraints and named defaults are not supported.

- **Creating data types**  In Sybase IQ, you can use the sp_addtype system
  procedure to add a domain, or you can use the CREATE DOMAIN
  statement. In Adaptive Server Enterprise, you must use sp_addtype.

### Data type conversions

**Description**

Type conversions happen automatically, or you can explicitly request them
using the CAST or CONVERT function.

If a string is used in a numeric expression or as an argument to a function
expecting a numeric argument, the string is converted to a number before use.

If a number is used in a string expression or as a string function argument, then
the number is converted to a string before use.

All date constants are specified as strings. The string is automatically
converted to a date before use.

There are certain cases where the automatic data type conversions are not
appropriate.

'12/31/90' + 5  -- Tries to convert the string to a number
'a' > 0  -- Tries to convert 'a' to a number

You can use the CAST or CONVERT function to force type conversions.

The following functions can also be used to force type conversions:
Data type conversions

- **DATE( expression )** – converts the expression into a date, and removes any hours, minutes or seconds. Conversion errors might be reported.

- **DATETIME( expression )** – converts the expression into a timestamp. Conversion errors might be reported.

- **STRING( expression )** – similar to `CAST(value AS CHAR)`, except that `STRING(NULL)` is the empty string (""), whereas `CAST(NULL AS CHAR)` is the NULL value.

For information about the `CAST` and `CONVERT` functions, see “Data type conversion functions” on page 108.

Compatibility of string to datetime conversions

There are some differences in behavior between Sybase IQ and Adaptive Server Enterprise when converting strings to date and time data types.

If you convert a string containing only a time value (no date) to a date/time data type, Sybase IQ and Adaptive Server Enterprise both use a default date of January 1, 1900. SQL Anywhere uses the current date.

If the milliseconds portion of a time is less than 3 digits, Adaptive Server Enterprise interprets the value differently depending on whether it was preceded by a period or a colon. If preceded by a colon, the value means thousandths of a second. If preceded by a period, 1 digit means tenths, 2 digits mean hundredths, and 3 digits mean thousandths. Sybase IQ and SQL Anywhere interpret the value the same way, regardless of the separator.

Example

- Adaptive Server Enterprise converts the values below as shown.
  
  12:34:56.7 to 12:34:56.700  
  12.34.56.78 to 12:34:56.780  
  12:34:56.789 to 12:34:56.789  
  12:34:56:7 to 12:34:56.007  
  12.34.56:78 to 12:34:56.078  
  12:34:56:789 to 12:34:56.789  

- Sybase IQ converts the milliseconds value in the manner that Adaptive Server Enterprise does for values preceded by a period, in both cases:
  
  12:34:56.7 to 12:34:56.700  
  12.34.56.78 to 12:34:56.780  
  12:34:56.789 to 12:34:56.789  
  12:34:56:7 to 12:34:56.007  
  12.34.56:78 to 12:34:56.078  
  12:34:56:789 to 12:34:56.789
Compatibility of exported dates

For dates in the first 9 days of a month and hours less than 10, Adaptive Server Enterprise supports a blank for the first digit; Sybase IQ supports a zero or a blank. For details on how to load such data from Adaptive Server Enterprise into Sybase IQ, see Chapter 7, “Moving Data In and Out of Databases” in System Administration Guide: Volume 1.

Conversion of BIT to BINARY data type

Sybase IQ supports BIT to BINARY and BIT to VARBINARY implicit and explicit conversion and is compatible with Adaptive Server Enterprise support of these conversions. Sybase IQ implicitly converts BIT to BINARY and BIT to VARBINARY data types for comparison operators, arithmetic operations, and INSERT and UPDATE statements.

For BIT to BINARY conversion, bit value ‘b’ is copied to the first byte of the binary string and the rest of the bytes are filled with zeros. For example, BIT value 1 is converted to BINARY(n) string 0x0100...00 having 2^n nibbles. BIT value 0 is converted to BINARY string 0x00...00.

For BIT to VARBINARY conversion, BIT value ‘b’ is copied to the first byte of the BINARY string and the remaining bytes are not used; that is, only one byte is used. For example, BIT value 1 is converted to VARBINARY(n) string 0x01 having 2 nibbles.

The result of both implicit and explicit conversions of BIT to BINARY and BIT to VARBINARY data types is the same. The following table contains examples of BIT to BINARY and VARBINARY conversions.

<table>
<thead>
<tr>
<th>Conversion of BIT value ‘1’ to</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY(3)</td>
<td>0x010000</td>
</tr>
<tr>
<td>VARBINARY(3)</td>
<td>0x01</td>
</tr>
<tr>
<td>BINARY(8)</td>
<td>0x0100000000000000</td>
</tr>
<tr>
<td>VARBINARY(8)</td>
<td>0x01</td>
</tr>
</tbody>
</table>

BIT to BINARY and BIT to VARBINARY conversion examples These examples illustrate both implicit and explicit conversion of BIT to BINARY and BIT to VARBINARY data types.

Given the following tables and data:

```
CREATE TABLE tbin(c1 BINARY(9))
CREATE TABLE tvarbin(c2 VARBINARY(9))
CREATE TABLE tbar(c2 BIT)

INSERT tbar VALUES(1)
INSERT tbar VALUES(0)
```

Implicit conversion of BIT to BINARY:
Data type conversions

INSERT tbin SELECT c2 FROM tbar

```
<table>
<thead>
<tr>
<th>c1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100000000000000000</td>
<td>(18 nibbles)</td>
</tr>
<tr>
<td>0x000000000000000000</td>
<td>(18 nibbles)</td>
</tr>
</tbody>
</table>
```

Implicit conversion of BIT to VARBINARY:

INSERT tvarbin SELECT c2 FROM tbar

```
<table>
<thead>
<tr>
<th>c2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td>0x00</td>
<td></td>
</tr>
</tbody>
</table>
```

Explicit conversion of BIT to BINARY:

INSERT tbin SELECT CONVERT (BINARY(9), c2) FROM tbar

```
<table>
<thead>
<tr>
<th>c1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100000000000000000</td>
<td>(18 nibbles)</td>
</tr>
<tr>
<td>0x000000000000000000</td>
<td>(18 nibbles)</td>
</tr>
</tbody>
</table>
```

Explicit conversion of BIT to VARBINARY:

INSERT tvarbin SELECT CONVERT(VARBINARY(9), c2) FROM tbar

```
<table>
<thead>
<tr>
<th>c2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td>0x00</td>
<td></td>
</tr>
</tbody>
</table>
```

Conversion between BIT and CHAR/VARCHAR data types

Sybase IQ supports implicit conversion between BIT and CHAR, and BIT and VARCHAR data types for comparison operators, arithmetic operations, and INSERT and UPDATE statements.

**BIT to VARCHAR, CHAR to BIT, and VARCHAR to BIT conversion examples**  These examples illustrate both implicit and explicit conversions between BIT and CHAR, and BIT and VARCHAR data types.

Given the following tables and data:

```
CREATE TABLE tchar(c1 CHAR(9))
CREATE TABLE tvarchar(c2 VARCHAR(9))
CREATE TABLE tbar(c2 BIT)
CREATE TABLE tbit(c2 BIT)
```
INSERT tbar VALUES(1)
INSERT tbar VALUES(0)

Implicit conversion of BIT to VARCHAR / VARCHAR to BIT and implicit conversion of BIT to VARCHAR:

INSERT tvarchar SELECT c2 FROM tbar
SELECT c2, char_length(c2) FROM tvarchar

<table>
<thead>
<tr>
<th>c2</th>
<th>char_length(tvarchar.c2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>'1'</td>
<td>1</td>
</tr>
<tr>
<td>'0'</td>
<td>1</td>
</tr>
</tbody>
</table>

Implicit conversion of VARCHAR to BIT:

INSERT tbit SELECT c2 FROM tvarchar
SELECT c2 FROM tbit

<table>
<thead>
<tr>
<th>c2</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Explicit conversion of BIT to CHAR / CHAR to BIT and explicit conversion of BIT to CHAR:

INSERT tchar SELECT CONVERT (CHAR(9), c2) FROM tbar
SELECT c1, char_length(c1) FROM tchar

<table>
<thead>
<tr>
<th>c1</th>
<th>char_length(tchar.c1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>'1'</td>
<td>9</td>
</tr>
<tr>
<td>'0'</td>
<td>9</td>
</tr>
</tbody>
</table>

Explicit conversion of CHAR to BIT:

INSERT tbit SELECT CONVERT (BIT, c1) FROM tchar
SELECT c2 FROM tbit

<table>
<thead>
<tr>
<th>c2</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Explicit conversion of BIT to VARCHAR / VARCHAR to BIT and explicit conversion of BIT to VARCHAR:

INSERT tvarchar SELECT CONVERT(VARCHAR(9), c2) FROM tbar
Sybase IQ binary load format

Description

Sybase IQ uses the FORMAT BINARY and BINARY column specification clauses to produce data files that can be read by the LOAD TABLE statement.

To speed data loading into Sybase IQ, customers can create data files in Sybase IQ binary format and load this data into Sybase IQ using the FORMAT BINARY syntax of the LOAD TABLE command.

You can find instructions for creating a load script using the LOAD TABLE syntax and specifying the load specification in Reference: Statements and Options.

Create data files with these binary formats to load into columns with the corresponding data types. In most cases, Sybase IQ uses the platform-specific binary format. These data types are exceptions that use binary formats that are specific to Sybase IQ:

- DATE
- TIME
- DATETIME
- NUMERIC

The Sybase IQ binary load format is a fixed width format.
In general, fixed width loads complete faster than variable width loads. When the load logic knows the length of a column and row, the data is processed more efficiently. Using delimiters to separate columns and rows that vary in width causes the load to spend time scanning the input data looking for them.

The IQ Binary Load Format is a fixed width load. The load can determine the width of each column and length of each row from information in the table definition.

**Note** Binary load format is endian-sensitive. This is because binary load format utilizes native binary data types to represent data.

<table>
<thead>
<tr>
<th>Operating system native data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT (1 byte)</td>
</tr>
<tr>
<td>TINYINT (1 byte)</td>
</tr>
<tr>
<td>SMALLINT (2 bytes)</td>
</tr>
<tr>
<td>INT/UNSIGNED INT (4 bytes)</td>
</tr>
<tr>
<td>BIGINT/UNSIGNED BIGINT (8 bytes)</td>
</tr>
<tr>
<td>FLOAT (4 bytes)</td>
</tr>
<tr>
<td>DOUBLE (8 bytes)</td>
</tr>
<tr>
<td>CHAR/VARCHAR (character data)</td>
</tr>
<tr>
<td>BINARY/VARBINARY (binary data)</td>
</tr>
</tbody>
</table>

Data for the following data types is stored in native operating system binary format and can be written to data files directly in that format. Sybase IQ reads the respective number of bytes directly into the associated data types without conversion.

- BIT (1 byte)
- TINYINT (1 byte)
- SMALLINT (2 bytes)
- INT/UNSIGNED INT (4 bytes)
- BIGINT/UNSIGNED BIGINT (8 bytes)
- FLOAT (4 bytes)
- DOUBLE (8 bytes)
- CHAR/VARCHAR (character data)
- BINARY/VARBINARY (binary data)

By default, VARCHAR and VARBINARY columns are read in as many bytes as specified by LOAD TABLE `column-spec`.

**DATE**

DATE column data is stored in Sybase IQ as four bytes (a 32-bit unsigned integer) representing the number of days since 0000-01-01. To convert a calendar date to the Sybase IQ binary format, use:

For a given year, month, and day:

```
year = current_year - 1;
days_in_year_0000 = 366;
binaryDateValue = (year * 365) + (year / 4) - (year / 100)
```
Sybase IQ binary load format

\[
\text{day}_0 = \text{days in year}_{0000} + \left(\frac{\text{year}}{400}\right) - 1;
\]

For the day\_of\_current\_year value in the formula above, consider the following example: February 12 is day 43.

**TIME**

TIME data is stored as a 64-bit unsigned quantity that represents a number in microseconds (in other words, 1.0e-6 seconds). The microsecond quantity is computed using:

For a given hour, minute, second, and microsecond (usec):

\[
\text{binaryTimeValue} = (\text{hour} \times 3600 + \text{minute} \times 60 + \text{second} + \text{microsecond}) \times 1000000
\]

**TIMESTAMP**

TIMESTAMP data is stored as a 64-bit unsigned integer and represents a quantity in microseconds. You can compute a binary TIMESTAMP value using:

For a given year, month, day, hour, minute, second, and microsecond:

- Compute \( \text{binaryDateValue} \) for the date as shown above.
- Compute \( \text{binaryTimeValue} \) for the time as shown above.

\[
\text{binaryDateTimeValue} = \text{binaryDateValue} \times 86400000000 + \text{binaryTimeValue}
\]

**NUMERIC and DECIMAL**

Formats for NUMERIC and DECIMAL data types vary as a function of precision. The value must be right-padded with zeroes to the full scale of the value. The value must also be fully left-padded with zeroes, but padding happens automatically with binary programming. Once the values are padded, the decimal point is removed. For example, the value 12.34 looks like:

- **NUMERIC(4,2):** 1234
- **NUMERIC(6,4):** 123400
- **NUMERIC(8,4):** 00123400
- **NUMERIC(12,6):** 000012340000
- **NUMERIC(16,8):** 0000001234000000

After the value is padded and the decimal point is removed, the following rules apply:

- If precision \(\leq 4\), then the binary format is identical to native operating system binary format for 2 byte integer quantity.
- If precision is between 5 and 9, then the binary format is identical to native operating system binary format for a 4 byte integer quantity.
If precision is between 10 and 18, then the binary format is identical to native operating system binary format for an 8 byte integer quantity.

If precision >= 19, then there is a special format that uses the following C++ struct definition:

```
struct {
    unsigned char sign; // sign 1 for +, 0 for -
    unsigned char ndig; // # digits
    unsigned char exp; // exponent
    unsigned short digits[80];
};
```

Exponent is excess-80 form, unless the value is zero. A “zero” value is represented as:

- sign = 1
- ndig = 0
- exp = 0

The maximum exponent value is 159. The maximum number of supported digits is 288. “digits[0]” contains the least-significant digits. Digits are stored in a packed representation with two digits per “unsigned short” (2-byte) quantity. For a given “digit:"

- lower order digit = digit[i] & 0x00FF
- high order digit => digit[i] & 0xFF00

For example, consider the value 100 loaded into a NUMERIC(20) column. The binary layout of this value is:

- Sign - 0x01
- Number digits - 0x01
- Exponent - 0x50
- Digits - 0x0064

As another example, consider the value 32769:

- Sign - 0x01
- Number digits - 0x02
- Exponent - 0x50
- Digits - 0x0ad1 0x0003

If you translate the digits into base 10, you have:

- 0x0ad1 => 2769
- 0x0003 => 3
Inserting NULL values is to use the NULL BYTE in the input file and specify WITH NULL BYTE in the column specification of the LOAD TABLE statement. This is done by terminating each data field in the input file with ‘x00’ or ‘x01’. Terminating a data field in the input file with ‘x01’ instructs the load to insert NULL into the column. For example:

```sql
CREATE TABLE d1 ( c1 DATE );
LOAD TABLE d1 ( c1 BINARY WITH NULL BYTE ) FROM 'filename' QUOTES OFF ESCAPES OFF FORMAT BINARY;
```

If the content of the load input file is 000b32cb00000b32cc00, two rows will be loaded to the table. The first row will be May 7, 2009 and the second May 8, 2009. Notice that a NULL BYTE was added to the input file after each binary date. If you want NULL loaded into the first row, change the value of the NULL BYTE in the input file to ‘x01’.

```sql
000b32cb01000b32cc00
```

The NULL portion of the column specification indicates how to treat certain input values as NULL values, when loading into the table column. These characters can include BLANKS, ZEROS, or any other list of literals you define. When you specify a NULL value or read a NULL value from the source file, the destination column must be able to contain NULLs.

ZEROS is interpreted as follows:

- The column is set to NULL if the input data is all binary zeros (not character zeros).
- If the input data is character zero:
  - NULL(ZEROS) never causes the column to be NULL.
  - NULL(‘0’) causes the column to be NULL. For example:

```sql
CREATE TABLE t1 ( c1 INT, c2 INT );
VIEW THE INPUT DATA FILE, WHICH USES BIG-ENDIAN BYTE ORDERING:
od -x data.inp
3030 3030 0000 04d2
EXECUTE:
LOAD TABLE t1 ( c1 ASCII(4) NULL( '0000' ),
c2 BINARY )
FROM 'data.inp'
FORMAT BINARY
QUOTES OFF
```
ESCAPES OFF;

The results:

```
SELECT * FROM t1;
c1  c2
NULL 1234
```

- If the input data is binary zero (all bits clear):
  - NULL(ZEROS) causes the column to be NULL.
  - NULL('0') never causes the column to be NULL, for example:

    Load:
    ```
    CREATE TABLE t1 ( c1 INT, c2 INT );
    VIEW the input data file, which uses big-endian byte ordering:
    od -x data.inp
    0000 0000 0000 04d2
    Execute:
    LOAD TABLE t1 ( c1 ASCII(4) NULL(zeros),
    c2 BINARY )
    FROM 'data.inp'
    FORMAT BINARY
    QUOTES OFF
    ESCAPES OFF;
    ```

    The results:
    ```
    SELECT * FROM t1;
c1  c2
NULL 1234
    ```

    As another example, if your LOAD TABLE statement includes `col1
date('yymmdd') null(zeros)` and the data to load is 000000, you
receive an error indicating that 000000 cannot be converted to a
DATE(4). To get LOAD TABLE to insert a NULL value in `col1` when the
data is 000000, either write the NULL clause as `null('000000')`, or
modify the data to equal binary zeros and use NULL(ZEROS).

Another way to load NULLs during a binary load is not to supply data for the
column in the LOAD TABLE statement, if the destination column accepts null
values. For example:

```
CREATE TABLE t1 ( c1 INT, c2 INT );
LOAD TABLE T1 ( c2 BINARY ) FROM 'data.inp'
FORMATE BINARY
```
Sybase IQ binary load format

```
QUOTES OFF
ESCAPES OFF;

SELECT * FROM T1;
c1    c2
NULL  1234
Null  1234
```

View the input data file, which uses big-endian byte ordering:
```
od -x data.inp
0000 04d2 0000 04d2
```
CHAPTER 4

SQL Functions

This chapter describes the built-in functions that Sybase IQ supports.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>102</td>
</tr>
<tr>
<td>Aggregate functions</td>
<td>102</td>
</tr>
<tr>
<td>Analytical functions</td>
<td>104</td>
</tr>
<tr>
<td>Data type conversion functions</td>
<td>108</td>
</tr>
<tr>
<td>Date and time functions</td>
<td>109</td>
</tr>
<tr>
<td>HTTP functions</td>
<td>114</td>
</tr>
<tr>
<td>Numeric functions</td>
<td>114</td>
</tr>
<tr>
<td>String functions</td>
<td>115</td>
</tr>
<tr>
<td>System functions</td>
<td>118</td>
</tr>
<tr>
<td>SQL and Java user-defined functions</td>
<td>122</td>
</tr>
<tr>
<td>Time series and forecasting functions</td>
<td>123</td>
</tr>
<tr>
<td>Miscellaneous functions</td>
<td>127</td>
</tr>
<tr>
<td>Alphabetical list of functions</td>
<td>128</td>
</tr>
</tbody>
</table>
Overview

Functions return information from the database and are allowed anywhere an expression is allowed.

When using functions with Sybase IQ:

- Unless otherwise stated, any function that receives the NULL value as a parameter returns a NULL value.
- If you omit the FROM clause, or if all tables in the query are in the SYSTEM dbspace, SQL Anywhere processes the query, instead of Sybase IQ, and might behave differently, especially with regard to syntactic and semantic restrictions and the effects of option settings. See the SQL Anywhere documentation for rules that might apply to processing.
- If you have a query that does not require a FROM clause, you can force Sybase IQ to process the query by adding the clause “FROM iq_dummy,” where iq_dummy is a one-row, one-column table that you create in your database.

Aggregate functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Aggregate functions summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>Simple aggregate functions, such as SUM(), MIN(), MAX(), AVG() and COUNT() are allowed only in the select list and in the HAVING and ORDER BY clauses of a SELECT statement. These functions summarize data over a group of rows from the database. Groups are formed using the GROUP BY clause of the SELECT statement. A new class of aggregate functions, called <strong>window functions</strong>, provides moving averages and cumulative measures that compute answers to queries such as, “What is the quarterly moving average of the Dow Jones Industrial average,” or “List all employees and their cumulative salaries for each department.”</td>
</tr>
<tr>
<td></td>
<td>• Simple aggregate functions, such as AVG(), COUNT(), MAX(), MIN(), and SUM() summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement.</td>
</tr>
</tbody>
</table>
• Newer statistical aggregate functions that take one argument include STDDEV(), STDDEV_SAMP(), STDDEV_POP(), VARIANCE(), VAR_SAMP(), and VAR_POP.

Both the simple and newer categories of aggregates can be used as a windowing function that incorporates a `<WINDOW CLAUSE>` in a SQL query specification (a window) that conceptually creates a moving window over a result set as it is processed. See “Analytical functions” on page 104.

Another class of window aggregate functions supports analysis of time series data. Like the simple aggregate and statistical aggregate functions, you can use these window aggregates in conjunction with a SQL query specification (or window-spec). The time series window aggregate functions calculate correlation, linear regression, ranking, and weighted average results:

• ANSI SQL:2008 OLAP functions for time series analysis include: CORR(), COVAR_POP(), COVAR_SAMP(), CUME_DIST(), FIRST_VALUE(), LAST_VALUE(), REGR_AVGX(), REGR_AVGY(), REGR_COUNT(), REGR_INTERCEPT(), REGR_R2(), REGR_SLOPE(), REGR_SXX(), REGR_SXY(), and REGR_SYY().

• Non-ANSI SQL:2008 OLAP aggregate function extensions used in the database industry include FIRST_VALUE(), MEDIAN(), and LAST_VALUE().

• Weighted OLAP aggregate functions that calculate weighted moving averages include EXP_WEIGHTED_AVG() and WEIGHTED_AVG().

<table>
<thead>
<tr>
<th>Table 4-1: Aggregate functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate function</td>
</tr>
<tr>
<td>AVG</td>
</tr>
<tr>
<td>CORR</td>
</tr>
<tr>
<td>COUNT</td>
</tr>
<tr>
<td>COUNT</td>
</tr>
<tr>
<td>COVAR_POP</td>
</tr>
<tr>
<td>COVAR_SAMP</td>
</tr>
<tr>
<td>CUME_DIST</td>
</tr>
<tr>
<td>EXP_WEIGHTED_AVG</td>
</tr>
<tr>
<td>FIRST_VALUE</td>
</tr>
<tr>
<td>LAST_VALUE</td>
</tr>
<tr>
<td>MAX</td>
</tr>
<tr>
<td>MEDIAN</td>
</tr>
<tr>
<td>MIN</td>
</tr>
<tr>
<td>REGR_AVGX</td>
</tr>
</tbody>
</table>
Analytical functions

Aggregate function | Parameters
--- | ---
REGR_AVGY | (dependent-expression, independent-expression)
REGR_COUNT | (dependent-expression, independent-expression)
REGR_INTERCEPT | (dependent-expression, independent-expression)
REGR_R2 | (dependent-expression, independent-expression)
REGR_SLOPE | (dependent-expression, independent-expression)
REGR_SXX | (dependent-expression, independent-expression)
REGR_SXY | (dependent-expression, independent-expression)
REGR_SYY | (dependent-expression, independent-expression)
STDDEV | ( [ ALL ] expression )
SUM | ( [ DISTINCT ] { column-name | numeric-expr } )
VARIANCE | ( [ ALL ] expression )
WEIGHTED_AVG | (expression, period-expression)

The aggregate functions AVG, SUM, STDDEV, and VARIANCE do not support the binary data types (BINARY and VARBINARY).

See also

See the individual analytical function descriptions in this chapter for specific details on the use of each function.

For more information about using OLAP functions, see Chapter 2, “Using OLAP” in the System Administration Guide: Volume 2.

Analytical functions

Function | Analytical functions include:
--- | ---
Simple aggregates — AVG, COUNT, MAX, MIN, and SUM, STDDEV, and VARIANCE

Note You can use all simple aggregates, except the Grouping() function, with an OLAP windowed function.

- Window functions:
  - Windowing aggregates — AVG, COUNT, MAX, MIN, and SUM.
  - Ranking functions — RANK, DENSE_RANK, PERCENT_RANK, and NTILE.
• Statistical functions — STDDEV, STDDEV_SAMP, STDDEV_POP, VARIANCE, VAR_SAMP, and VAR_POP.

• Distribution functions — PERCENTILE_CONT and PERCENTILE_DISC.

• Numeric functions — WIDTH_BUCKET, CEIL, and LN, EXP, POWER, SQRT, and FLOOR.

Windowing aggregate function usage

A major feature of the ANSI SQL extensions for OLAP is a construct called a window. This windowing extension lets users divide result sets of a query (or a logical partition of a query) into groups of rows called partitions and determine subsets of rows to aggregate with respect to the current row.

You can use three classes of window functions with a window: ranking functions, the row numbering function, and window aggregate functions.

Windowing extensions specify a window function type over a window name or specification and are applied to partitioned result sets within the scope of a single query expression. A window partition is a subset of rows returned by a query, as defined by one or more columns in a special OVER clause:

OVER (PARTITION BY col1, col2...)

Windowing operations let you establish information such as the ranking of each row within its partition, the distribution of values in rows within a partition, and similar operations. Windowing also lets you compute moving averages and sums on your data, enhancing the ability to evaluate your data and its impact on your operations.

A window partition is a subset of rows returned by a query, as defined by one or more columns in a special OVER() clause:

OVER (PARTITION BY col1, col2...)

Ranking functions usage

The OLAP ranking functions let application developers compose single-statement SQL queries that answer questions such as “Name the top 10 products shipped this year by total sales,” or “Give the top 5% of salespeople who sold orders to at least 15 different companies.” These functions include the ranking functions, RANK(), DENSE_RANK(), PERCENT_RANK() and NTILE() with a PARTITION BY clause.

Rank analytical functions rank items in a group, compute distribution, and divide a result set into a number of groupings. The rank analytical functions, RANK, DENSE_RANK, PERCENT_RANK, and NTILE all require an OVER (ORDER BY) clause. For example:

RANK() OVER ( [PARTITION BY] ORDER BY <expression> [ ASC | DESC ] )
Analytical functions

The ORDER BY clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the OVER clause and is not an ORDER BY for SELECT. No aggregation functions in the rank query are allowed to specify DISTINCT.

The OVER clause indicates that the function operates on a query result set. The result set is the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have all been evaluated. The OVER clause defines the data set of the rows to include in the computation of the rank analytical function.

The value expression is a sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

The ASC or DESC parameter specifies the ordering sequence as ascending or descending. Ascending order is the default.

Rank analytical functions are only allowed in the select list of a SELECT or INSERT statement or in the ORDER BY clause of the SELECT statement. Rank functions can be in a view or a union. You cannot use rank functions in a subquery, a HAVING clause, or in the select list of an UPDATE or DELETE statement. More than one rank analytical function is allowed per query in Sybase IQ 15.1.

Statistical aggregate analytic function usage

Summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement. Aggregate functions are allowed only in the select list and in the HAVING and ORDER BY clauses of a SELECT statement. These functions include STDDEV, STDDEV_POP, STDDEV_SAMP, VARIANCE, VAR_POP, and VAR_SAMP.

The OLAP functions can be used as a window function with an OVER() clause in a SQL query specification that conceptually creates a moving window over a result set as it is processed.

Distribution functions usage

The inverse distribution analytical functions PERCENTILE_CONT and PERCENTILE_DISC take a percentile value as the function argument and operate on a group of data specified in the WITHIN GROUP clause, or operate on the entire data set. These functions return one value per group. For PERCENTILE_DISC, the data type of the results is the same as the data type of its ORDER BY item specified in the WITHIN GROUP clause. For PERCENTILE_CONT, the data type of the results is either numeric, if the ORDER BY item in the WITHIN GROUP clause is a numeric, or double, if the ORDER BY item is an integer or floating-point.

The inverse distribution analytical functions require a WITHIN GROUP (ORDER BY) clause. For example:
PERCENTILE_CONT (expression1) WITHIN GROUP (ORDER BY expression2 [ASC | DESC])

The value of expression1 must be a constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a “wrong argument for percentile” error is returned. If the argument value is less than 0, or greater than 1, then a “data value out of range” error is returned.

The ORDER BY clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the WITHIN GROUP clause and is not an ORDER BY for the SELECT.

The WITHIN GROUP clause distributes the query result into an ordered data set from which the function calculates a result.

The value expression2 is a sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

The ASC or DESC parameter specifies the ordering sequence as ascending or descending. Ascending order is the default.

Inverse distribution analytical functions are allowed in a subquery, a HAVING clause, a view, or a union. The inverse distribution functions can be used anywhere the simple non analytical aggregate functions are used. The inverse distribution functions ignore the NULL value in the data set.

Table 4-2 lists the analytical functions and their parameters. Unlike aggregate functions in Table 4-1, you cannot specify DISTINCT in window functions.

**Table 4-2: Analytical functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>( { column-name</td>
</tr>
<tr>
<td>COUNT</td>
<td>( * )</td>
</tr>
<tr>
<td>COUNT</td>
<td>( { column-name</td>
</tr>
<tr>
<td>DENSE_RANK</td>
<td>()</td>
</tr>
<tr>
<td>GROUPING *</td>
<td>( { GROUPING group-by-expression } )</td>
</tr>
<tr>
<td>MAX</td>
<td>( { column-name</td>
</tr>
<tr>
<td>MIN</td>
<td>( { column-name</td>
</tr>
<tr>
<td>NTILE</td>
<td>( integer )</td>
</tr>
<tr>
<td>PERCENT_RANK</td>
<td>()</td>
</tr>
<tr>
<td>PERCENTILE_CONT</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>PERCENTILE_DISC</td>
<td>( numeric-expr )</td>
</tr>
</tbody>
</table>
**Data type conversion functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANK</td>
<td>()</td>
</tr>
<tr>
<td>STDDEV</td>
<td>([ ALL ] expression)</td>
</tr>
<tr>
<td>STDDEV_POP</td>
<td>([ ALL ] expression)</td>
</tr>
<tr>
<td>STDDEV_SAMP</td>
<td>([ ALL ] expression)</td>
</tr>
<tr>
<td>SUM</td>
<td>( { column-name</td>
</tr>
<tr>
<td>VAR_POP</td>
<td>([ ALL ] expression)</td>
</tr>
<tr>
<td>VAR_SAMP</td>
<td>([ ALL ] expression)</td>
</tr>
<tr>
<td>VARIANCE</td>
<td>([ ALL ] expression)</td>
</tr>
</tbody>
</table>

* The OLAP SQL standard allows Grouping() in GROUP BY CUBE, or GROUP BY ROLLUP operations only.

**Compatibility**

The ranking and inverse distribution analytical functions are not supported by Adaptive Server Enterprise.

**See also**

See the individual analytical function descriptions in this chapter for specific details on the use of each function.


**Data type conversion functions**

Data type conversion functions convert arguments from one data type to another.

Table 4-3 lists the data type conversion functions and their parameters.
Table 4-3: Date type conversion functions

<table>
<thead>
<tr>
<th>Data type conversion function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINTTOHEX</td>
<td>(integer-expression)</td>
</tr>
<tr>
<td>CAST</td>
<td>(expression AS data type)</td>
</tr>
<tr>
<td>CONVERT</td>
<td>(data type, expression [, format-style])</td>
</tr>
<tr>
<td>HEXTOBIGINT</td>
<td>(hexadecimal-string)</td>
</tr>
<tr>
<td>HEXTOINT</td>
<td>(hexadecimal-string)</td>
</tr>
<tr>
<td>INTTOHEX</td>
<td>(integer-expr)</td>
</tr>
<tr>
<td>ISDATE</td>
<td>(string)</td>
</tr>
<tr>
<td>ISNUMERIC</td>
<td>(string)</td>
</tr>
</tbody>
</table>

Description

The DATE, DATETIME, DATEFORMAT, and YMD functions that convert expressions to dates, timestamps, or strings based on a date format are listed in “Date and time functions” on page 109. The STRING function, which converts expressions to a string, is discussed in the section “String functions” on page 115.

The database server carries out many type conversions automatically. For example, if a string is supplied where a numerical expression is required, the string is automatically converted to a number. For more information on automatic data type conversions carried out by Sybase IQ, see “Data type conversions” on page 89.

Date and time functions

Date and time functions perform conversion, extraction, or manipulation operations on date and time data types and can return date and time information.

Table 4-4 and Table 4-5 list the date and time functions and their parameters.
## Date and time functions

### Syntax 1

<table>
<thead>
<tr>
<th>Date and time functions</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>(expression)</td>
</tr>
<tr>
<td>DATECEILING</td>
<td>(date-part, datetime-expr, [multiple-expr])</td>
</tr>
<tr>
<td>DATEFLOOR</td>
<td>(date-part, datetime-expr, [multiple-expr])</td>
</tr>
<tr>
<td>DATEFORMAT</td>
<td>(datetime-expr, string-expr)</td>
</tr>
<tr>
<td>DATENAME</td>
<td>(date-part, date-expr)</td>
</tr>
<tr>
<td>DATEROUND</td>
<td>(date-part, datetime-expr, [multiple-expr])</td>
</tr>
<tr>
<td>DATETIME</td>
<td>(expression)</td>
</tr>
<tr>
<td>DAY</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>DAYNAME</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>DAYS</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>DAYS</td>
<td>(date-expr, date-expr)</td>
</tr>
<tr>
<td>DAYS</td>
<td>(date-expr, integer-expr)</td>
</tr>
<tr>
<td>DOW</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>HOUR</td>
<td>(datetime-expr)</td>
</tr>
<tr>
<td>HOURS</td>
<td>(datetime-expr)</td>
</tr>
<tr>
<td>HOURS</td>
<td>(datetime-expr, datetime-expr)</td>
</tr>
<tr>
<td>HOURS</td>
<td>(datetime-expr, integer-expr)</td>
</tr>
<tr>
<td>ISDATE</td>
<td>(string)</td>
</tr>
<tr>
<td>MINUTE</td>
<td>(datetime-expr)</td>
</tr>
<tr>
<td>MINUTES</td>
<td>(datetime-expr)</td>
</tr>
<tr>
<td>MINUTES</td>
<td>(datetime-expr, datetime-expr)</td>
</tr>
<tr>
<td>MINUTES</td>
<td>(datetime-expr, integer-expr)</td>
</tr>
<tr>
<td>MONTH</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>MONTHNAME</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>MONTHS</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>MONTHS</td>
<td>(date-expr, date-expr)</td>
</tr>
<tr>
<td>MONTHS</td>
<td>(date-expr, integer-expr)</td>
</tr>
<tr>
<td>NOW</td>
<td>(* *)</td>
</tr>
<tr>
<td>QUARTER</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>SECOND</td>
<td>(datetime-expr)</td>
</tr>
<tr>
<td>SECONDS</td>
<td>(datetime-expr)</td>
</tr>
<tr>
<td>SECONDS</td>
<td>(datetime-expr, datetime-expr)</td>
</tr>
<tr>
<td>SECONDS</td>
<td>(datetime-expr, integer-expr)</td>
</tr>
<tr>
<td>TODAY</td>
<td>(* *)</td>
</tr>
<tr>
<td>WEEKS</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>WEEKS</td>
<td>(date-expr, date-expr)</td>
</tr>
</tbody>
</table>
CHAPTER 4  SQL Functions

Reference: Building Blocks, Tables, and Procedures

Table 4-5: Transact-SQL-compatible date and time functions

<table>
<thead>
<tr>
<th>Date and time functions</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEEKS</td>
<td>(date-expr, integer-expr)</td>
</tr>
<tr>
<td>YEAR</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>YEARS</td>
<td>(date-expr)</td>
</tr>
<tr>
<td>YEARS</td>
<td>(date-expr, date-expr)</td>
</tr>
<tr>
<td>YEARS</td>
<td>(date-expr, integer-expr)</td>
</tr>
<tr>
<td>YMD</td>
<td>(year-num, month-num, day-num)</td>
</tr>
</tbody>
</table>

SybaseIQ provides two classes of date and time functions that can be used interchangeably, but have different styles. One set is Transact-SQL-compatible.

The date and time functions listed in Table 4-4 allow manipulation of time units. Most time units (such as MONTH) have four functions for time manipulation, although only two names are used (such as MONTH and MONTHS).

The functions listed in Table 4-5 are the Transact-SQL date and time functions. They allow an alternative way of accessing and manipulating date and time functions.

You should convert arguments to date functions to dates before used them. For example, this is incorrect:

    days ( '1995-11-17', 2 )

This is correct:

    days ( date( '1995-11-17' ), 2 )

Sybase IQ does not have the same constants or data type promotions as SQL Anywhere, with which it shares a common user interface. If you issue a SELECT statement without a FROM clause, the statement is passed to SQL Anywhere. The following statement is handled exclusively by SQL Anywhere:
SELECT WEEKS('1998/11/01');

The following statement, processed by Sybase IQ, uses a different starting point for the WEEKS function and returns a different result than the statement above:

SELECT WEEKS('1998/11/01') FROM iq_dummy;

Consider another example. The MONTHS function returns the number of months since an “arbitrary starting date.” The “arbitrary starting date” of Sybase IQ, the imaginary date 0000-01-01, is chosen to produce the most efficient date calculations and is consistent across various data parts. SQL Anywhere does not have a single starting date. The following statements, the first processed by SQL Anywhere, the second by Sybase IQ, both return the answer 12:

SELECT MONTHS('0001/01/01');
SELECT MONTHS('0001/01/01') FROM iq_dummy;

However, also consider these statements:

SELECT DAYS('0001/01/01');
SELECT DAYS('0001/01/01') FROM iq_dummy;

The first, processed by SQL Anywhere, yields the value 307, but the second, processed by Sybase IQ, yields 166.

For the most consistent results, therefore, always include the table name in the FROM clause whether you need it or not.

Note Create a dummy table with only one column and row. You can then reference this table in the FROM clause for any SELECT statement that uses date or time functions, thus ensuring processing by Sybase IQ, and consistent results.

Date parts

Many of the date functions use dates built from date parts. Table 4-6 displays allowed values of date-part.
### Table 4-6: Date part values

<table>
<thead>
<tr>
<th>Date part</th>
<th>Abbreviation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>yy</td>
<td>0001 – 9999</td>
</tr>
<tr>
<td>Quarter</td>
<td>qq</td>
<td>1 – 4</td>
</tr>
<tr>
<td>Month</td>
<td>mm</td>
<td>1 – 12</td>
</tr>
<tr>
<td>Week</td>
<td>wk</td>
<td>1 – 54</td>
</tr>
<tr>
<td>Day</td>
<td>dd</td>
<td>1 – 31</td>
</tr>
<tr>
<td>Dayofyear</td>
<td>dy</td>
<td>1 – 366</td>
</tr>
<tr>
<td>Weekday</td>
<td>dw</td>
<td>1 – 7 (Sun. – Sat.)</td>
</tr>
<tr>
<td>Hour</td>
<td>hh</td>
<td>0 – 23</td>
</tr>
<tr>
<td>Minute</td>
<td>mi</td>
<td>0 – 59</td>
</tr>
<tr>
<td>Second</td>
<td>ss</td>
<td>0 – 59</td>
</tr>
<tr>
<td>Millisecond</td>
<td>ms</td>
<td>0 – 999</td>
</tr>
<tr>
<td>Calyearofweek</td>
<td>cyr</td>
<td>Integer. The year in which the week begins. The week containing the first few days of the year can be part of the last week of the previous year, depending upon which day it begins. If the new year starts on a Thursday through Saturday, its first week starts on the last Sunday of the previous year. If the new year starts on a Sunday through Wednesday, none of its days are part of the previous year.</td>
</tr>
<tr>
<td>Calweekofyear</td>
<td>cwk</td>
<td>An integer from 1 to 54 representing the week number within the year that contains the specified date.</td>
</tr>
<tr>
<td>Caldowofweek</td>
<td>cdw</td>
<td>The day number within the week (Sunday = 1, Saturday = 7).</td>
</tr>
</tbody>
</table>

**Note** By default, Sunday is the first day of the week. To make Monday be the first day, use:

```sql
set option ‘Date_First_Day_Of_Week’ = ‘1’
```

For more information on specifying which day is the first day of the week, see DATE_FIRST_DAY_OF_WEEK option in *Reference: Statements and Options*.

**Compatibility**

For compatibility with Adaptive Server Enterprise, use the Transact-SQL date and time functions.
HTTP functions

Function

HTTP functions facilitate the handling of HTTP requests within Web services.

Table 4-7 lists all HTTP functions and their parameters.

<table>
<thead>
<tr>
<th>HTTP function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML_DECODE</td>
<td>(string)</td>
</tr>
<tr>
<td>HTML_ENCODE</td>
<td>(string)</td>
</tr>
<tr>
<td>HTTP_DECODE</td>
<td>(string)</td>
</tr>
<tr>
<td>HTTP_ENCODE</td>
<td>(string)</td>
</tr>
<tr>
<td>HTTP_VARIABLE</td>
<td>(var-name [ , instance ], header-field )</td>
</tr>
<tr>
<td>NEXT_HTTP_HEADER</td>
<td>header-name</td>
</tr>
<tr>
<td>NEXT_HTTP_VARIABLE</td>
<td>var-name</td>
</tr>
</tbody>
</table>

Numeric functions

Function

Numeric functions perform mathematical operations on numerical data types or return numeric information.

Sybase IQ does not have the same constants or data type promotions as SQL Anywhere, with which it shares a common user interface. If you issue a SELECT statement without a FROM clause, the statement is passed through to SQL Anywhere. For the most consistent results, include the table name in the FROM clause whether you need it or not.

Note Consider creating a dummy table to use in such cases.

Table 4-8 lists numeric functions and their parameters.
## Table 4-8: Numeric functions

<table>
<thead>
<tr>
<th>Numeric function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>ACOS</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>ASIN</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>ATAN</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>ATAN2</td>
<td>( numeric-expr1, numeric-expr2 )</td>
</tr>
<tr>
<td>CEIL</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>CEILING</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>COS</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>COT</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>DEGREES</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>EXP</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>FLOOR</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>LN</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>LOG</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>LOG10</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>MOD</td>
<td>( dividend, divisor )</td>
</tr>
<tr>
<td>PI</td>
<td>( * )</td>
</tr>
<tr>
<td>POWER</td>
<td>( numeric-expr1, numeric-expr2 )</td>
</tr>
<tr>
<td>RADIANS</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>RAND</td>
<td>( ( integer-expr ) )</td>
</tr>
<tr>
<td>REMAINDER</td>
<td>( numeric-expr, numeric-expr )</td>
</tr>
<tr>
<td>ROUND</td>
<td>( numeric-expr, integer-expr )</td>
</tr>
<tr>
<td>SIGN</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>SIN</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>SQRT</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>SQUARE</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>TAN</td>
<td>( numeric-expr )</td>
</tr>
<tr>
<td>“TRUNCATE”</td>
<td>( numeric-expr, integer-expr )</td>
</tr>
<tr>
<td>TRUNCNUM</td>
<td>( numeric-expression, integer-expression )</td>
</tr>
<tr>
<td>WIDTH_BUCKET</td>
<td>( expression, min_value, max_value, num_buckets )</td>
</tr>
</tbody>
</table>

### String functions

Reference: Building Blocks, Tables, and Procedures 115
Function

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

When working in a multibyte character set, check carefully whether the function being used returns information concerning characters or bytes.

Most of the string functions accept binary data (hexadecimal strings) in the `string-expr` parameter, but some of the functions, such as `LCASE`, `UCASE`, `LOWER`, and `LTRIM`, expect the string expression to be a character string.

Unless you supply a constant `LENGTH` argument to a function that produces a `VARCHAR` result (such as `SPACE` or `REPEAT`), the default length is the maximum allowed. See the “Field Size” column in Table 6-1 on page 348.

Sybase IQ queries containing one or more of these functions might return one of the following errors:

ASA Error -1009080: Key doesn't fit on a single database page: 65560(4, 1)

ASA Error -1009119: Record size too large for database page size

For example:

```sql
SELECT COUNT(*) FROM test1 a WHERE (a.col1 + SPACE(4-LENGTH(a.col1)) + a.col2 + space(2-LENGTH(a.col2))) IN (SELECT (b.col3) FROM test1 b);
```

To avoid such errors, cast the function result with an appropriate maximum length; for example:

```sql
SELECT COUNT(*) FROM test1 a WHERE (a.col1 + CAST(SPACE(4-LENGTH(a.col1)) AS VARCHAR(4)) + a.col2 + CAST(SPACE(2-LENGTH(a.col2)) AS VARCHAR(4))) IN (SELECT (b.col3) FROM test1 b);
```

The errors are more likely with an IQ page size of 64K or a multibyte collation.

Table 4-9 lists string functions and their parameters.
<table>
<thead>
<tr>
<th>String function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>BIT_LENGTH</td>
<td>(column-name)</td>
</tr>
<tr>
<td>BYTE_LENGTH</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>CHAR</td>
<td>(integer-expr)</td>
</tr>
<tr>
<td>CHAR_LENGTH</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>CHARINDEX</td>
<td>(string-expr1, string-expr2)</td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>(string-expr1, string-expr2)</td>
</tr>
<tr>
<td>GRAPHICAL_PLAN</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>HTML_PLAN</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>INSERTSTR</td>
<td>(numeric-expr, string-expr1, string-expr2)</td>
</tr>
<tr>
<td>LCASE</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>LEFT</td>
<td>(string-expr, numeric-expr)</td>
</tr>
<tr>
<td>LEN</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>LENGTH</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>LOCATE</td>
<td>(string-expr1, string-expr2 [ , numeric-expr ])</td>
</tr>
<tr>
<td>LOWER</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>LTRIM</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>OCTET_LENGTH</td>
<td>(column-name)</td>
</tr>
<tr>
<td>PATINDEX</td>
<td>(%pattern%, string_expr)</td>
</tr>
<tr>
<td>REPEAT</td>
<td>(string-expr, numeric-expr)</td>
</tr>
<tr>
<td>REPLACE</td>
<td>(original-string, search-string, replace-string)</td>
</tr>
<tr>
<td>REVERSE</td>
<td>(expression</td>
</tr>
<tr>
<td>REPLICATE</td>
<td>(string-expr, integer-expr)</td>
</tr>
<tr>
<td>RIGHT</td>
<td>(string-expr, numeric-expr)</td>
</tr>
<tr>
<td>RTRIM</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>SIMILAR</td>
<td>(string-expr1, string-expr2)</td>
</tr>
<tr>
<td>SORTKEY</td>
<td>(string-expression</td>
</tr>
<tr>
<td></td>
<td>[, {collation-id</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SOUNDEX</td>
<td>(string-expr)</td>
</tr>
<tr>
<td>SPACE</td>
<td>(integer-expr)</td>
</tr>
<tr>
<td>STR</td>
<td>(numeric_expr [, length [ , decimal ]] )</td>
</tr>
<tr>
<td>STR_REPLACE</td>
<td>(string_expr1, string_expr2, string_expr3)</td>
</tr>
<tr>
<td>STRING</td>
<td>(string1 [, string2, ..., string99 ] )</td>
</tr>
<tr>
<td>STUFF</td>
<td>(string-expr1, start, length, string-expr2)</td>
</tr>
</tbody>
</table>
## System functions

System functions return system information.

Table 4-10 lists the system functions and their parameters.

<table>
<thead>
<tr>
<th>String function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBSTRING</td>
<td>( string-expr, integer-expr [, integer-expr ] )</td>
</tr>
<tr>
<td>TRIM</td>
<td>( string-expr )</td>
</tr>
<tr>
<td>UCASE</td>
<td>( string-expr )</td>
</tr>
<tr>
<td>UPPER</td>
<td>( string-expr )</td>
</tr>
</tbody>
</table>
### Table 4-10: System functions

<table>
<thead>
<tr>
<th>System function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL_LENGTH</td>
<td>(table-name, column-name)</td>
</tr>
<tr>
<td>COL_NAME</td>
<td>(table-id, column-id [ , database-id ])</td>
</tr>
<tr>
<td>CONNECTION_PROPERTY</td>
<td>( { property-id</td>
</tr>
<tr>
<td>DATALength</td>
<td>(expression)</td>
</tr>
<tr>
<td>DB_ID</td>
<td>([database-name])</td>
</tr>
<tr>
<td>DB_NAME</td>
<td>([database-id])</td>
</tr>
<tr>
<td>DB_PROPERTY</td>
<td>( { property-id</td>
</tr>
<tr>
<td>EVENT_CONDITION</td>
<td>(condition-name)</td>
</tr>
<tr>
<td>EVENT_CONDITION_NAME</td>
<td>(integer)</td>
</tr>
<tr>
<td>EVENT_PARAMETER</td>
<td>(context-name)</td>
</tr>
<tr>
<td>GROUP_MEMBER</td>
<td>(group-name-string-expression [ , user-name-string-expression ])</td>
</tr>
<tr>
<td>INDEX_COL</td>
<td>(table-name, index-id, key_# [ ,user-id ])</td>
</tr>
<tr>
<td>NEXT_CONNECTION</td>
<td>([connection-id] [ , database-id])</td>
</tr>
<tr>
<td>NEXT_DATABASE</td>
<td>([NULL</td>
</tr>
<tr>
<td>OBJECT_ID</td>
<td>(object-name)</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>(object-id [ , database-id])</td>
</tr>
<tr>
<td>PROPERTY</td>
<td>( { property-id</td>
</tr>
<tr>
<td>PROPERTY_DESCRIPTION</td>
<td>(property-id</td>
</tr>
<tr>
<td>PROPERTY_NAME</td>
<td>(property-id)</td>
</tr>
<tr>
<td>PROPERTY_NUMBER</td>
<td>(property-name)</td>
</tr>
<tr>
<td>SUSER_ID</td>
<td>([user-name])</td>
</tr>
<tr>
<td>SUSER_NAME</td>
<td>([user-id])</td>
</tr>
<tr>
<td>USER_ID</td>
<td>([user-name])</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>([user-id])</td>
</tr>
</tbody>
</table>

### Description

Databases currently running on a server are identified by a database name and a database ID number. The `db_id` and `db_name` functions provide information on these values.

A set of system functions provides information about properties of a currently running database, or of a connection, on the database server. These system functions take the database name or ID, or the connection name, as an optional argument to identify the database or connection for which the property is requested.
System functions

Performance

System functions are processed differently than other Sybase IQ functions. When queries to Sybase IQ tables include system functions, performance is reduced.

Compatibility

Table 4-11 shows the Adaptive Server Enterprise system functions and their status in Sybase IQ:

<table>
<thead>
<tr>
<th>Function</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>col_length</td>
<td>Implemented</td>
</tr>
<tr>
<td>col_name</td>
<td>Implemented</td>
</tr>
<tr>
<td>db_id</td>
<td>Implemented</td>
</tr>
<tr>
<td>db_name</td>
<td>Implemented</td>
</tr>
<tr>
<td>index_col</td>
<td>Implemented</td>
</tr>
<tr>
<td>object_id</td>
<td>Implemented</td>
</tr>
<tr>
<td>object_name</td>
<td>Implemented</td>
</tr>
<tr>
<td>proc_role</td>
<td>Always returns 0</td>
</tr>
<tr>
<td>show_role</td>
<td>Always returns NULL</td>
</tr>
<tr>
<td>tsequal</td>
<td>Not implemented</td>
</tr>
<tr>
<td>user_id</td>
<td>Implemented</td>
</tr>
<tr>
<td>user_name</td>
<td>Implemented</td>
</tr>
<tr>
<td>suser_id</td>
<td>Implemented</td>
</tr>
<tr>
<td>suser_name</td>
<td>Implemented</td>
</tr>
<tr>
<td>datalength</td>
<td>Implemented</td>
</tr>
<tr>
<td>curunreservedpgs</td>
<td>Not implemented</td>
</tr>
<tr>
<td>data_pgs</td>
<td>Not implemented</td>
</tr>
<tr>
<td>host_id</td>
<td>Not implemented</td>
</tr>
<tr>
<td>host_name</td>
<td>Not implemented</td>
</tr>
<tr>
<td>lct_admin</td>
<td>Not implemented</td>
</tr>
<tr>
<td>reserved_pgs</td>
<td>Not implemented</td>
</tr>
<tr>
<td>rowcnt</td>
<td>Not implemented</td>
</tr>
<tr>
<td>used_pgs</td>
<td>Not implemented</td>
</tr>
<tr>
<td>valid_name</td>
<td>Not implemented</td>
</tr>
<tr>
<td>valid_user</td>
<td>Not implemented</td>
</tr>
</tbody>
</table>

Notes

- Some of the system functions are implemented in Sybase IQ as system stored procedures.
- The db_id, db_name, datalength, suser_id, and suser_name functions are implemented as built-in functions.
CHAPTER 4  SQL Functions

Connection properties

Connection properties apply to an individual connection. This section describes how to retrieve the value of a specific connection property or the values of all connection properties. For descriptions of all connection properties, see “Connection properties” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Configuring Your Database > Connection, database, and database server properties.

Examples

❖ Retrieving the value of a connection property
  • Use the connection_property system function. The following statement returns the number of pages that have been read from file by the current connection:

        select connection_property ( 'DiskRead' )

❖ Retrieving the values of all connection properties
  • Use the sa_conn_properties system procedure:

        call sa_conn_properties

A separate row appears for each connection, for each property.

Properties available for the server

Server properties apply across the server as a whole. This section describes how to retrieve the value of a specific server property or the values of all server properties. For descriptions of all server properties, see “Database server properties” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Configuring Your Database > Connection, database, and database server properties.

Examples

❖ Retrieving the value of a server property
  • Use the property system function. The following statement returns the number of cache pages being used to hold the main heap:

        select property ( 'MainHeapPages') from iq_dummy

❖ Retrieving the values of all server properties
  • Use the sa_eng_properties system procedure:
Properties available for each database

Database properties apply to an entire database. This section describes how to retrieve the value of a specific database property or the values of all database properties. For descriptions of all of the database properties, see “Database server properties” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Configuring Your Database > Connection, database, and database server properties.

Examples

❖ Retrieving the value of a database property
  • Use the db_property system function. The following statement returns the page size of the current database:

    \[
    \text{select db_property ('PageSize') from iq_dummy}
    \]

❖ Retrieving the values of all database properties
  • Use the sa_db_properties system procedure:

    \[
    \text{call sa_db_properties}
    \]

SQL and Java user-defined functions

There are two mechanisms for creating user-defined functions in Sybase IQ. You can use the SQL language to write the function, or you can use Java.

Note  User-defined functions are processed by SQL Anywhere. They do not take advantage of the performance features of Sybase IQ. Queries that include user-defined functions run at least 10 times slower than queries without them.

In very few cases, differences in semantics between SQL Anywhere and Sybase IQ can produce different results for a query if it is issued in a user-defined function. For example, Sybase IQ treats the CHAR and VARCHAR data types as distinct and different, while SQL Anywhere treats CHAR data as if it were VARCHAR.
User-defined functions in SQL

You can implement your own functions in SQL using the `CREATE FUNCTION` statement. The `RETURN` statement inside the `CREATE FUNCTION` statement determines the data type of the function.

Once you have created a SQL user-defined function, you can use it anywhere a built-in function of the same data type is used.

**Note** Avoid using the CONTAINS predicate in a view that has a user-defined function, because the CONTAINS criteria is ignored. Use the LIKE predicate instead, or issue the query outside of a view.

For more information on creating SQL functions, see Chapter 1, “Using Procedures and Batches” in the *System Administration Guide: Volume 2*.

User-defined functions in Java

Although SQL functions are useful, Java classes provide a more powerful and flexible way of implementing user-defined functions, with the additional advantage that you can move them from the database server to a client application if desired.

Any **class method** of an installed Java class can be used as a user-defined function anywhere a built-in function of the same data type is used.

Instance methods are tied to particular instances of a class, and so have different behavior from standard user-defined functions.

For more information on creating Java classes, and on class methods, see “Java support in SQL Anywhere” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - Programming > Java in the database*.

**Time series and forecasting functions**

Time series and forecasting functions are aggregate functions (see “Aggregate functions” on page 102) designed exclusively for financial time series analysis.

**Note** Time series and forecasting capability is available only with RAP – The Trading Edition™ Enterprise.
Loading the IMSL libraries for time series and forecasting functions

The time series and forecasting functions call two third-party external libraries. The IMSL™ C Stat and C Math libraries, provided by Visual Numerics Inc., contain C functions used for financial time series and forecasting calculations.

The wrapper library libtsudf contains user-defined functions (UDFs) that invoke functions contained in the IMSL C Stat and C Math libraries.

### Table 4-12: Time series functions

<table>
<thead>
<tr>
<th>Time series functions</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS_ARMA_AR</td>
<td>(timeseries_expression, ar_count, ar_elem, method)</td>
</tr>
<tr>
<td>TS_ARMA_CONST</td>
<td>(timeseries_expression, method)</td>
</tr>
<tr>
<td>TS_ARMA_MA</td>
<td>(timeseries_expression, ma_count, ma_elem, method)</td>
</tr>
<tr>
<td>TS_AUTO_CORRELATION</td>
<td>(timeseries_expression, lagmax, lag_elem)</td>
</tr>
<tr>
<td>TS_AUTO_UNI_AR</td>
<td>(timeseries_expression, ar_count, ar_elem, method)</td>
</tr>
<tr>
<td>TS_BOX_COX_XFORM</td>
<td>(timeseries_expression, power [, shift [, inverse]])</td>
</tr>
<tr>
<td>TS_DIFFERENCE</td>
<td>(timeseries_expression, period1 [, period2 [, ...period 10]])</td>
</tr>
<tr>
<td>TS_ESTIMATE_MISSING</td>
<td>(timeseries_expression, method)</td>
</tr>
<tr>
<td>TS_LACK_OF_FIT</td>
<td>(timeseries_expression, p_value, q_value, lagmax, [tolerance])</td>
</tr>
<tr>
<td>TS_LACK_OF_FIT_P</td>
<td>(timeseries_expression, p_value, q_value, lagmax, [tolerance])</td>
</tr>
<tr>
<td>TS_MAX_ARMA_AR</td>
<td>(timeseries_expression, ar_count, ar_elem)</td>
</tr>
<tr>
<td>TS_MAX_ARMA_CONST</td>
<td>(timeseries_expression)</td>
</tr>
<tr>
<td>TS_MAX_ARMA_MA</td>
<td>(timeseries_expression, ma_count, ma_elem)</td>
</tr>
<tr>
<td>TS_MAX_ARMA_LIKELIHOOD</td>
<td>(timeseries_expression)</td>
</tr>
<tr>
<td>TS_OUTLIER_IDENTIFICATION</td>
<td>(timeseries_expression, p_value, q_value, s_value, d_value, [, delta_value[, critical_value]])</td>
</tr>
<tr>
<td>TS_PARTIAL_AUTO_CORRELATION</td>
<td>(timeseries_expression, lagmax, lag_elem)</td>
</tr>
<tr>
<td>TS_VWAP</td>
<td>(price_expression, volume_expression)</td>
</tr>
</tbody>
</table>
The time series and forecasting SQL functions call the `libtsudf` wrapper library automatically. Sybase IQ loads the IMSL C Stat and C Math libraries when you call a valid user-defined aggregate function for time series and forecasting analysis.

The names and locations of the IMSL C Stat and C Math libraries vary depending on the platform on which Sybase IQ is installed:

<table>
<thead>
<tr>
<th>Directory where libraries are located</th>
<th>Windows 32-bit</th>
<th>Windows 64-bit</th>
<th>UNIX (except AIX)</th>
<th>AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bin32</td>
<td>bin64</td>
<td>lib64</td>
<td>lib64</td>
</tr>
<tr>
<td>Library file name</td>
<td>imslcstat_imsl_dll.dll</td>
<td>imslcstat_imsl_dll.dll</td>
<td>libimslcstat_imsl.so</td>
<td>libimslcstat_imsl_r.so</td>
</tr>
<tr>
<td></td>
<td>imslcmath_imsl_dll.dll</td>
<td>imslcmath_imsl_dll.dll</td>
<td>libimslcmath_imsl.so</td>
<td>libimslcmath_imsl_r.so</td>
</tr>
</tbody>
</table>

**IMSL library time series function error-handling**

You can control the error-handling behavior for the time series functions that call the IMSL libraries. If a runtime error occurs when invoking IMSL library functions, Sybase IQ responds according to your error-handling choice. You can choose from four error-handling options, ranging from ignoring all errors and warnings (the default behavior), to specifying which level of error severity causes Sybase IQ to abort a SQL statement and return an error message.

Use the following SQL statement to control error-handling:

```
set option PUBLIC.Time_Series_Error_Level = '<value>'
```

Valid values are:
Table 4-14: IMSL library time series function error-handling

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (default)</td>
<td>All types of warnings and errors that can be obtained while invoking an IMSL library function are ignored. When such a condition is encountered, the time series function returns a NULL value.</td>
</tr>
<tr>
<td>1</td>
<td>If the time series function obtains a warning or an error message while invoking an IMSL library function, IQ returns an error message and aborts the SQL query.</td>
</tr>
<tr>
<td>2</td>
<td>If the time series function obtains a fatal error message while invoking an IMSL library function, IQ returns an error message and aborts the SQL query. However if a warning is obtained then the time series function returns a NULL value.</td>
</tr>
<tr>
<td>3</td>
<td>If the time series function obtains a terminal error message while invoking an IMSL library function, IQ returns an error message and aborts the SQL query. However if a warning or a fatal error is obtained then the time series function returns a NULL value.</td>
</tr>
</tbody>
</table>

IMSL library time series function error logging

You can control how Sybase IQ logs the error messages returned by IMSL library time series function error-handling. Four options exist for logging error messages to the log file.

Use the following SQL statement to control error logging:

```
set option PUBLIC.Time_Series_Log_Level = '<value>'
```
### Table 4-15: IMSL library time series function error logging

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (default)</td>
<td>All warnings and errors returned while invoking an IMSL library function are ignored and not logged to the log file.</td>
</tr>
<tr>
<td>1</td>
<td>If the time series function returns a warning or an error message while invoking an IMSL library function, a message is logged to the log file.</td>
</tr>
<tr>
<td>2</td>
<td>If the time series function returns a fatal error message while invoking an IMSL library function, a message is logged to the log file. Warnings are not logged.</td>
</tr>
<tr>
<td>3</td>
<td>If the time series function returns a terminal error message while invoking an IMSL library function, a message is logged to the log file. Warnings and fatal errors are not logged.</td>
</tr>
</tbody>
</table>

### Miscellaneous functions

**Function**

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

Table 4-16 lists the miscellaneous functions and their parameters.

**Table 4-16: Miscellaneous functions**

<table>
<thead>
<tr>
<th>Miscellaneous functions</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARGN</td>
<td>(integer-expr, expression [ , … ] )</td>
</tr>
<tr>
<td>COALESCE</td>
<td>(expression, expression [ , expression ... ] )</td>
</tr>
<tr>
<td>IFNULL</td>
<td>(expression1, expression2 [ , expression3 ] )</td>
</tr>
<tr>
<td>ISNULL</td>
<td>(expression, expression [ , expression ... ] )</td>
</tr>
<tr>
<td>NULLIF</td>
<td>(expression1, expression2 )</td>
</tr>
<tr>
<td>NUMBER</td>
<td>( * )</td>
</tr>
<tr>
<td>ROWID</td>
<td>(table-name )</td>
</tr>
</tbody>
</table>

**Compatibility**

Adaptive Server Enterprise supports only the COALESCE, ISNULL, and NULLIF functions.
Alphabetical list of functions

This section describes each function individually. The function type, for example, Numeric or String, is indicated in brackets next to the function name. Some of the results in the examples have been rounded or truncated. The actual values of database object IDs, such as the object ID of a table or the column ID of a column, might differ from the values shown in the examples.

ABS function [Numeric]

Function
Returns the absolute value of a numeric expression.

Syntax
ABS ( numeric-expression )

Parameters
numeric-expression  The number whose absolute value is to be returned.

Example
The following statement returns the value 66:

```
SELECT ABS( -66 ) FROM iq_dummy
```

Standards and compatibility
• SQL92  Vendor extension
• Sybase  Compatible with Adaptive Server Enterprise

ACOS function [Numeric]

Function
Returns the arc-cosine, in radians, of a numeric expression.

Syntax
ACOS ( numeric-expression )

Parameters
numeric-expression  The cosine of the angle.

Example
The following statement returns the value 1.023945:

```
SELECT ACOS( 0.52 ) FROM iq_dummy
```

Standards and compatibility
• SQL92  Vendor extension
• Sybase  Compatible with Adaptive Server Enterprise

See also
“ASIN function [Numeric]” on page 129
“ATAN function [Numeric]” on page 130
“ATAN2 function [Numeric]” on page 130
“COS function [Numeric]” on page 143
ARGN function [Miscellaneous]

Function
Returns a selected argument from a list of arguments.

Syntax
\[ \text{ARGN}( \text{integer-expression}, \text{expression}, [...]) \]

Parameters
- \text{integer-expression} The position of an argument within a list of expressions.
- \text{expression} An expression of any data type passed into the function. All supplied expressions must be of the same data type.

Example
The following statement returns the value 6:
\[ \text{SELECT ARGN( 6, 1,2,3,4,5,6) FROM iq_dummy} \]

Usage
Using the value of \text{integer-expression} as \( n \) returns the \( n \)th argument (starting at 1) from the remaining list of arguments. While the expressions can be of any data type, they must all be of the same data type. The integer expression must be from one to the number of expressions in the list or NULL is returned. Multiple expressions are separated by a comma.

Standards and compatibility
- SQL92 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise

ASCII function [String]

Function
Returns the integer ASCII value of the first byte in a string-expression.

Syntax
\[ \text{ASCII}( \text{string-expression}) \]

Parameters
- \text{string-expression} The string

Example
The following statement returns the value 90, when the collation sequence is set to the default ISO_BINENG:
\[ \text{SELECT ASCII( 'Z') FROM iq_dummy} \]

Usage
If the string is empty, ASCII returns zero. Literal strings must be enclosed in quotes.

Standards and compatibility
- SQL92 Vendor extension
- Sybase Compatible with Adaptive Server Enterprise

ASIN function [Numeric]

Function
Returns the arc-sine, in radians, of a number.
ASIN function

**Syntax**

`ASIN ( numeric-expression )`

**Parameters**

- **numeric-expression**: The sine of the angle

**Example**

The following statement returns the value 0.546850:

```
SELECT ASIN( 0.52 ) FROM iq_dummy
```

**Standards and compatibility**

- **SQL92**: Vendor extension
- **Sybase**: Compatible with Adaptive Server Enterprise

**See also**

- “ACOS function [Numeric]” on page 128
- “ATAN function [Numeric]” on page 130
- “ATAN2 function [Numeric]” on page 130
- “SIN function [Numeric]” on page 250

ATAN function

**Function**

Returns the arc-tangent, in radians, of a number.

**Syntax**

`ATAN ( numeric-expression )`

**Parameters**

- **numeric-expression**: The tangent of the angle

**Example**

The following statement returns the value 0.479519:

```
SELECT ATAN( 0.52 ) FROM iq_dummy
```

**Standards and compatibility**

- **SQL92**: Vendor extension
- **Sybase**: Compatible with Adaptive Server Enterprise

**See also**

- “ACOS function [Numeric]” on page 128
- “ASIN function [Numeric]” on page 129
- “ATAN function [Numeric]” on page 130
- “TAN function [Numeric]” on page 267

ATAN2 function

**Function**

Returns the arc-tangent, in radians, of the ratio of two numbers.

**Syntax**

`ATAN2 ( numeric-expression1, numeric-expression2 )`
Parameters

**numeric-expression1**  The numerator in the ratio whose arc tangent is calculated.

**numeric-expression2**  The denominator in the ratio whose arc-tangent is calculated.

Example

The following statement returns the value 0.00866644968879073143:

```
SELECT ATAN2( 0.52, 0.60 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  ATAN2 is not supported by Adaptive Server Enterprise

See also

“ACOS function [Numeric]” on page 128

“ASIN function [Numeric]” on page 129

“ATAN function [Numeric]” on page 130

“TAN function [Numeric]” on page 267

---

**AVG function [Aggregate]**

Function

Computes the average of a numeric expression for a set of rows, or computes the average of a set of unique values.

Syntax

```
AVG ( numeric-expression | DISTINCT column-name )
```

Parameters

**numeric-expression**  The value whose average is calculated over a set of rows.

**DISTINCT column-name**  Computes the average of the unique values in `column-name`. This is of limited usefulness, but is included for completeness.

Example

The following statement returns the value 49988.6:

```
SELECT AVG ( salary ) FROM Employees
```

Usage

This average does not include rows where `numeric-expression` is the NULL value. Returns the NULL value for a group containing no rows.

Standards and compatibility

- **SQL92**  SQL92 compatible.
- **Sybase**  Compatible with Adaptive Server Enterprise.

See also

“COUNT function [Aggregate]” on page 146

“SUM function [Aggregate]” on page 265

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*
BIGINTTOHEX function [Data type conversion]

Function
Returns the hexadecimal equivalent in VARCHAR(16) of a decimal integer.

Syntax
BIGINTTOHEX ( integer-expression )

Parameters
integer-expression  The integer to be converted to hexadecimal.

Examples
The following statement returns the value 0000000000000009:
SELECT BIGINTTOHEX(9) FROM iq_dummy

The following statement returns the value FFFFFFFFFFFFFF7:
SELECT BIGINTTOHEX(-9) FROM iq_dummy

Usage
The BIGINTTOHEX function accepts an integer expression that evaluates to BIGINT and returns the hexadecimal equivalent. Returned values are left appended with zeros up to a maximum of 16 digits. All types of unscaled integer data types are accepted as integer expressions.

Conversion is done automatically, if required. Constants are truncated, only if the fraction values are zero. A column cannot be truncated, if the column is declared with a positive scale value. If conversion fails, Sybase IQ returns an error unless the CONVERSION_ERROR option is OFF. In that case, the result is NULL.

Standards and compatibility
• SQL92  Transact-SQL extension
• Sybase  Compatible with Adaptive Server Enterprise

See also
CONVERSION_ERROR option [TSQL] in Reference: Statements and Options
“HEXTOBIGINT function [Data type conversion]” on page 178
“HEXTOINT function [Data type conversion]” on page 179
“INTTOHEX function [Data type conversion]” on page 189

BIT_LENGTH function [String]

Function
Returns an unsigned 64-bit value containing the bit length of the column parameter.

Syntax
BIT_LENGTH( column-name )

Parameters
column-name  The name of a column

Usage
The return value of a NULL argument is NULL.
The `BIT_LENGTH` function supports all Sybase IQ data types.

**Standards and compatibility**

- **Sybase**: Not supported by SQL Anywhere or Adaptive Server Enterprise

**See also**

- "OCTET_LENGTH function [String]" on page 216

---

**BYTE_LENGTH function [String]**

**Function**

Returns the number of bytes in a string.

**Syntax**

```
BYTE_LENGTH ( string-expression )
```

**Parameters**

- **string-expression**: The string whose length is to be calculated

**Example**

The following statement returns the value 12:

```
SELECT BYTE_LENGTH( 'Test Message' ) FROM iq_dummy
```

**Usage**

- Trailing white space characters are included in the length returned.
- The return value of a NULL string is NULL.
- If the string is in a multibyte character set, the `BYTE_LENGTH` value differs from the number of characters returned by `CHAR_LENGTH`.

**Standards and compatibility**

- **SQL92**: Vendor extension
- **Sybase**: Not supported by Adaptive Server Enterprise

**See also**

- "CHAR_LENGTH function [String]" on page 136
- "DATALENGTH function [System]" on page 148
- "LENGTH function [String]" on page 197

---

**CAST function [Data type conversion]**

**Function**

Returns the value of an expression converted to a supplied data type.

**Syntax**

```
CAST ( expression AS data type )
```

**Parameters**

- **expression**: The expression to be converted
- **data type**: The target data type

**Examples**

The following function ensures a string is used as a date:

```
CAST( '2000-10-31' AS DATE )
```
Alphabetical list of functions

The value of the expression \(1 + 2\) is calculated, and the result cast into a single-character string, the length the data server assigns:

\[
\text{CAST}( \ 1 + 2 \ \text{AS} \ \text{CHAR} )
\]

You can use the \text{CAST} function to shorten strings:

\[
\text{SELECT CAST( lname AS CHAR(5) ) FROM Customers}
\]

Usage

If you do not indicate a length for character string types, Sybase IQ chooses an appropriate length. If neither precision nor scale is specified for a DECIMAL conversion, the database server selects appropriate values.

If neither precision nor scale is specified for the explicit conversion of NULL to \text{NUMERIC}, the default is \text{NUMERIC}(1,0). For example,

\[
\text{SELECT CAST( NULL AS NUMERIC ) A,}
\text{CAST( NULL AS NUMERIC(15,2) ) B}
\]

is described as:

\[
A \ \text{NUMERIC}(1,0)
\]
\[
B \ \text{NUMERIC}(15,2)
\]

Standards and compatibility

- SQL92 This function is SQL92 compatible.
- Sybase Compatible with Adaptive Server Enterprise.

See also

“\text{CONVERT} function [Data type conversion]” on page 139

\section*{CEIL function [Numeric]}

Function

Returns the smallest integer greater than or equal to the specified expression. \text{CEIL} is as synonym for \text{CEILING}.

Syntax

\[
\text{CEIL( numeric-expression )}
\]

Parameters

expression A column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, \text{CEIL} generates an error. The return value has the same data type as the value supplied.

Usage

For a given expression, the \text{CEIL} function takes one argument. For example, \text{CEIL(-123.45)} returns -123. \text{CEIL(123.45)} returns 124.

Standards and compatibility

- SQL92 Vendor extension
- Sybase Compatible with Adaptive Server Enterprise

See also

“\text{CEILING} function [Numeric]” on page 135
CEILING function [Numeric]

Function

Returns the ceiling (smallest integer not less than) of a number.

CEIL is as synonym for CEILING.

Syntax

CEILING ( numeric-expression )

Parameters

numeric-expression The number whose ceiling is to be calculated

Examples

The following statement returns the value 60.00000:

SELECT CEILING( 59.84567 ) FROM iq_dummy

The following statement returns the value 123:

SELECT CEILING( 123 ) FROM iq_dummy

The following statement returns the value 124.00:

SELECT CEILING( 123.45 ) FROM iq_dummy

The following statement returns the value -123.00:

SELECT CEILING( -123.45 ) FROM iq_dummy

Standards and compatibility

• SQL92 Vendor extension

• Sybase Compatible with Adaptive Server Enterprise

See also

“FLOOR function [Numeric]” on page 173

CHAR function [String]

Function

Returns the character with the ASCII value of a number.

Syntax

CHAR ( integer-expression )

Parameters

integer-expression The number to be converted to an ASCII character. The number must be in the range 0 to 255, inclusive.

Examples

The following statement returns the value “Y”:

SELECT CHAR( 89 ) FROM iq_dummy

The following statement returns the value “S”:

SELECT CHAR( 83 ) FROM iq_dummy
Usage

The character in the current database character set corresponding to the supplied numeric expression modulo 256 is returned.

CHAR returns NULL for integer expressions with values greater than 255 or less than zero.

Standards and compatibility

• SQL92 Vendor extension
• Sybase Compatible with Adaptive Server Enterprise

CHAR_LENGTH function [String]

Function

Returns the number of characters in a string.

Syntax

CHAR_LENGTH ( string-expression )

Parameters

string-expression The string whose length is to be calculated

Usage

Trailing white space characters are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the CHAR_LENGTH value may be less than the BYTE_LENGTH value.

Example

The following statement returns the value 8:

```
SELECT CHAR_LENGTH( 'Chemical' ) FROM iq_dummy
```

Standards and compatibility

• SQL92 This function is SQL92 compatible
• Sybase Compatible with Adaptive Server Enterprise

See also

“BYTE_LENGTH function [String]” on page 133

CHARINDEX function [String]

Function

Returns the position of the first occurrence of a specified string in another string.

Syntax

CHARINDEX ( string-expression1, string-expression2 )

Parameters

string-expression1 The string for which you are searching. This string is limited to 255 bytes.

string-expression2 The string to be searched. The position of the first character in the string being searched is 1.

Example

The statement:
SELECT Surname, GivenName
FROM Employees
WHERE CHARINDEX('K', Surname ) = 1

returns the following values:

<table>
<thead>
<tr>
<th>Surname</th>
<th>GivenName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klobucher</td>
<td>James</td>
</tr>
<tr>
<td>Kuo</td>
<td>Felicia</td>
</tr>
<tr>
<td>Kelly</td>
<td>Moira</td>
</tr>
</tbody>
</table>

Usage
All the positions or offsets, returned or specified, in the CHARINDEX function are always character offsets and may be different from the byte offset for multibyte data.

If the string being searched contains more than one instance of the specified string, CHARINDEX returns the position of the first instance.

If the string being searched does not contain the specified string, CHARINDEX returns zero (0).

Searching for a zero-length string returns 1.

If any of the arguments is NULL, the result is NULL.

CHARINDEX returns a 32 bit signed integer position for CHAR and VARCHAR columns.

Standards and compatibility
- SQL92 Vendor extension
- Sybase Compatible with Adaptive Server Enterprise

See also
“SUBSTRING function [String]” on page 265
Chapter 4, “Function Support” in Large Objects Management in Sybase IQ

COALESCE function [Miscellaneous]
Function
Returns the first non-NULL expression from a list.

Syntax
COALESCE ( expression, expression [, ... ] )

Parameters
expression Any expression

Example
The following statement returns the value 34:

```
SELECT COALESCE( NULL, 34, 13, 0 ) FROM iq_dummy
```

Standards and compatibility
- SQL92 SQL92
• **Sybase**  Compatible with Adaptive Server Enterprise.

### COL_LENGTH function [System]

**Function**
Returns the defined length of a column.

**Syntax**
```
COL_LENGTH ( table-name, column-name )
```

**Parameters**
- `table-name`: The table name
- `column-name`: The column name

**Example**
The following statement returns the column length 35:
```
SELECT COL_LENGTH ( 'CUSTOMERS', 'ADDRESS' ) FROM iq_dummy
```

### COL_NAME function [System]

**Function**
Returns the column name.

**Syntax**
```
COL_NAME ( table-id, column-id [ , database-id ] )
```

**Parameters**
- `table-id`: The object ID of the table
- `column-id`: The column ID of the column
- `database-id`: The database ID

**Examples**
The following statement returns the column name `lname`. The object ID of the Customers table is 100209, as returned by the `OBJECT_ID` function. The column ID is stored in the `column_id` column of the `syscolumn` system table. The database ID of the `iqdemo` database is 0, as returned by the `DB_ID` function.
```
SELECT COL_NAME( 100209, 3, 0 ) FROM iq_dummy
```
The following statement returns the column name `city`.
```
SELECT COL_NAME ( 100209, 5 ) FROM iq_dummy
```

### Standards and compatibility
- **SQL92**  Vendor extension
- **Sybase**  Adaptive Server Enterprise function implemented for Sybase IQ

See also
“DATALENGTH function [System]” on page 148
See also

“DB_ID function [System]” on page 161
“OBJECT_ID function [System]” on page 215

**CONNECTION_PROPERTY function [System]**

**Function**
Returns the value of a given connection property as a string.

**Syntax**
```
CONNECTION_PROPERTY ( { integer-expression1 | string-expression } 
... [ , integer-expression2 ] )
```

**Note**

**Parameters**
- **integer-expression1** In most cases, it is more convenient to supply a string expression as the first argument. If you do supply `integer-expression1`, it is the connection property ID. You can determine this using the `PROPERTY_NUMBER` function.
- **string-expression** The connection property name. You must specify either the property ID or the property name.
- **integer-expression2** The connection ID of the current database connection. The current connection is used if this argument is omitted.

**Example**
The following statement returns the number of prepared statements being maintained, for example, 4:
```
SELECT connection_property( 'PrepStmt' ) FROM iq_dummy
```

**Usage**
The current connection is used if the second argument is omitted.

**Standards and compatibility**
- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

**See also**
“Connection properties” on page 121
“PROPERTY_NUMBER function [System]” on page 227

**CONVERT function [Data type conversion]**

**Function**
Returns an expression converted to a supplied data type.

**Syntax**
```
CONVERT ( data-type, expression [, format-style ] )
```
Alphabetical list of functions

Parameters

data-type  The data type to which the expression is converted

expression  The expression to be converted

format-style  For converting strings to date or time data types and vice versa, format-style is a style code number that describes the date format string to be used. Table 4-17 lists the meanings of the values of the format-style argument.

<table>
<thead>
<tr>
<th>Without century (yy)</th>
<th>With century (yyyy)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0 or 100</td>
<td>mmm dd yyyy hh:mmAM (or PM)</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>mm/dd/yy[yy]</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>[yy]yy:mm.dd</td>
</tr>
<tr>
<td>3</td>
<td>103</td>
<td>dd/mm/yyyy[yy]</td>
</tr>
<tr>
<td>4</td>
<td>104</td>
<td>dd.mm.yy[yy]</td>
</tr>
<tr>
<td>5</td>
<td>105</td>
<td>dd-mm-yy[yy]</td>
</tr>
<tr>
<td>6</td>
<td>106</td>
<td>dd mm[yy][yy]</td>
</tr>
<tr>
<td>7</td>
<td>107</td>
<td>mmm dd, yy[yy]</td>
</tr>
<tr>
<td>8</td>
<td>108</td>
<td>hh:mm:ss</td>
</tr>
<tr>
<td>-</td>
<td>9 or 109</td>
<td>mmm dd yyyy hh:mm:ssAM (or PM)</td>
</tr>
<tr>
<td>10</td>
<td>110</td>
<td>mm-dd-yy[yy]</td>
</tr>
<tr>
<td>11</td>
<td>111</td>
<td>[yy]yy/mm/dd</td>
</tr>
<tr>
<td>12</td>
<td>112</td>
<td>[yy]ymd</td>
</tr>
<tr>
<td>13</td>
<td>113</td>
<td>dd mm[yy] yyyy hh:mm:ss (24 hour clock, Europe default + milliseconds, 4-digit year)</td>
</tr>
<tr>
<td>14</td>
<td>114</td>
<td>hh:mm:ss (24 hour clock)</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>yyyy-mm-dd hh:mm:ss (24-hour clock, ODBC canonical, 4-digit year)</td>
</tr>
<tr>
<td>21</td>
<td>121</td>
<td>yyyy-mm-dd hh:mm:ss.sss (24 hour clock, ODBC canonical with milliseconds, 4-digit year)</td>
</tr>
<tr>
<td>-</td>
<td>365</td>
<td>yyyyjjj (as a string or integer, where jjj is the Julian day number from 1 to 366 within the year)</td>
</tr>
</tbody>
</table>

If no format-style argument is provided, style code 0 is used.

Examples

The following statements illustrate the use of format styles:

SELECT CONVERT( CHAR( 20 ), order_date, 104 )
FROM sales_order

order_date
16.03.1993
CHAPTER 4  SQL Functions

**order_date**

20.03.1993
23.03.1993
25.03.1993
...

```
SELECT CONVERT( CHAR(20), order_date, 7 )
FROM sales_order
```

**order_date**

mar 16, 93
mar 20, 93
mar 23, 93
mar 25, 93
...

The following statements illustrate the use of the format style 365, which converts data of type *DATE* and *DATETIME* to and from either string or integer type data:

```
CREATE TABLE tab
    (date_col DATE, int_col INT, char7_col CHAR(7));
INSERT INTO tab (date_col, int_col, char7_col)
VALUES (‘Dec 17, 2004’, 2004352, ‘2004352’);
SELECT CONVERT(VARCHAR(8), tab.date_col, 365) FROM tab;
returns ‘2004352’
SELECT CONVERT(INT, tab.date_col, 365) from tab;
returns 2004352
SELECT CONVERT(DATE, tab.int_col, 365) FROM TAB;
returns 2004-12-17
SELECT CONVERT(DATE, tab.char7_col, 365) FROM tab;
returns 2004-12-17
```

The following statement illustrates conversion to an integer, and returns the value 5.

```
SELECT CONVERT( integer, 5.2 ) FROM iq_dummy
```

**Usage**

The result data type of a `CONVERT` function is a `LONG VARCHAR`. If you use `CONVERT` in a `SELECT INTO` statement, you must have a Large Objects Management option license or use `CAST` and set `CONVERT` to the correct data type and size.
Alphabetical list of functions

See “REPLACE function [String]” for more information.

Standards and compatibility

- SQL92 Vendor extension
- Sybase Compatible with Adaptive Server Enterprise and SQL Anywhere, except for format style 365, which is an IQ-only extension

See also “CAST function [Data type conversion]” on page 133

**CORR function [Aggregate]**

Function

Returns the correlation coefficient of a set of number pairs.

Syntax 1

```
CORR (dependent-expression, independent-expression)
```

Syntax 2

```
CORR (dependent-expression, independent-expression)
OVER (window-spec)
```

*window-spec*: See Syntax 2 instructions in the Usage section, below.

Parameters

- **dependent-expression** The variable that is affected by the independent-expression.
- **independent-expression** The variable that influences the outcome.

Example

The following example performs a correlation to discover whether age is associated with income level. This function returns the value 0.440227:

```
SELECT CORR( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) ) FROM Employees;
```

Usage

The CORR function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then CORR returns NULL.

**dependent-expression** and **independent-expression** are both numeric. The function is applied to the set of (**dependent-expression**, **independent-expression**) after eliminating the pairs for which either **dependent-expression** or **independent-expression** is NULL. The following computation is made:

```
COVAR_POP (y, x) / (STDDEV_POP (x) * STDDEV_POP (y))
```

where x represents the **dependent-expression** and y represents the **independent-expression**.

**Note** ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1.
Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Standards and compatibility

- **SQL2008**  SQL foundation feature outside of core SQL
- **Sybase**  Compatible with SQL Anywhere

### COS function [Numeric]

**Function**

Returns the cosine of a number, expressed in radians.

**Syntax**

```
COS ( numeric-expression )
```

**Parameters**

- `numeric-expression`  The angle, in radians.

**Example**

The following statement returns the value 0.86781:

```
SELECT COS( 0.52 ) FROM iq_dummy
```

**Standards and compatibility**

- **SQL92**  Vendor extension
- **Sybase**  Compatible with Adaptive Server Enterprise

**See also**

- “ACOS function [Numeric]” on page 128
- “COT function [Numeric]” on page 143
- “SIN function [Numeric]” on page 250
- “TAN function [Numeric]” on page 267

### COT function [Numeric]

**Function**

Returns the cotangent of a number, expressed in radians.

**Syntax**

```
COT ( numeric-expression )
```

**Parameters**

- `numeric-expression`  The angle, in radians.

**Example**

The following statement returns the value 1.74653:

```
SELECT COT( 0.52 ) FROM iq_dummy
```

**Standards and compatibility**

- **SQL92**  Vendor extension
- **Sybase**  Compatible with Adaptive Server Enterprise

**See also**

- “COS function [Numeric]” on page 143
**COVAR_POP function [Aggregate]**

**Function**

Returns the population covariance of a set of number pairs.

**Syntax 1**

\[
\text{COVAR\_POP}(\text{dependent-expression}, \text{independent-expression})
\]

**Syntax 2**

\[
\text{COVAR\_POP}(\text{dependent-expression}, \text{independent-expression}) \quad \text{OVER} \quad (\text{window-spec})
\]

*window-spec*: See Syntax 2 instructions in the Usage section, below.

**Parameters**

- **dependent-expression**: The variable that is affected by the independent variable.
- **independent-expression**: The variable that influences the outcome.

**Example**

The following example measures the strength of association between employee age and salary. This function returns the value 73785.84059:

\[
\text{SELECT COVAR\_POP( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) ) FROM Employees;}
\]

**Usage**

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then COVAR\_POP returns NULL.

**dependent-expression** and **independent-expression** are both numeric. The function is applied to the set of \((\text{dependent-expression}, \text{independent-expression})\) after eliminating the pairs for which either **dependent-expression** or **independent-expression** is NULL. The following computation is made:

\[
\frac{\text{SUM}(x*y) - \text{SUM}(x) \times \text{SUM}(y) / n}{n}
\]

where \(x\) represents the **dependent-expression** and \(y\) represents the **independent-expression**.

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions.
Note

ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Standards and compatibility

- SQL2008 SQL foundation feature outside of core SQL
- Sybase Compatible with SQL Anywhere

COVAR_SAMP function [Aggregate]

Function

Returns the sample covariance of a set of number pairs.

Syntax 1

```
COVAR_SAMP (dependent-expression, independent-expression)
```

Syntax 2

```
COVAR_SAMP (dependent-expression, independent-expression)
OVER (window-spec)
```

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

- **dependent-expression** The variable that is affected by the independent variable.
- **independent-expression** The variable that influences the outcome.

Example

The following example measures the strength of association between employee age and salary. This function returns the value 74782.946005:

```
SELECT COVAR_SAMP ( Salary, ( 2008 - YEAR ( BirthDate ) ) )
FROM Employees;
```

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then COVAR_SAMP returns NULL.

Both dependent-expression and independent-expression are numeric. The function is applied to the set of (dependent-expression, independent-expression) after eliminating the pairs for which either dependent-expression or independent-expression is NULL.

```
(SUM(x*y) - SUM(x) * SUM(y) / n) / (n-1)
```
where $x$ represents the `dependent-expression` and $y$ represents the `independent-expression`.

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions.

**Note** ROLLUP and CUBE are not supported in the `GROUP BY` clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a `SELECT` statement. As such, you can specify elements of `window-spec` either in the function syntax (inline), or with a `WINDOW` clause in the `SELECT` statement. For information on how to specify the window, see “Analytical functions” on page 104.

### Standards and compatibility
- **SQL2008** SQL foundation feature outside of core SQL
- **Sybase** Compatible with SQL Anywhere

### COUNT function [Aggregate]

**Function**
Counts the number of rows in a group, depending on the specified parameters.

**Syntax**
```
COUNT ( * | expression | DISTINCT column-name )
```

**Parameters**
- `*` Returns the number of rows in each group.
- `expression` Returns the number of rows in each group where `expression` is not the NULL value.
- `DISTINCT column-name` Returns the number of different values in `column-name`. Rows where the value is the NULL value are not included in the count.

**Example**
The following statement returns each unique city, and the number of rows with that city value:
```
SELECT city , Count(*)
FROM Employees
GROUP BY city
```

**Standards and compatibility**
- **SQL92** SQL92 compatible.
- **Sybase** Compatible with Adaptive Server Enterprise.

**See also**
- “AVG function [Aggregate]” on page 131
- “SUM function [Aggregate]” on page 265
CUME_DIST function [Ranking]

Function

The CUME_DIST function calculates the relative position of one value among a group of rows. It returns a decimal value between 0 and 1.

Syntax

CUME_DIST() OVER (window-spec)

window-spec: See the Usage section below.

Example

The following example returns a result set that provides a cumulative distribution of the salaries of employees who live in California:

```sql
SELECT DepartmentID, Surname, Salary, CUME_DIST() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) "Rank"
FROM Employees
WHERE State IN ('CA');
```

The returned result set is:

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>Surname</th>
<th>Salary</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Savarino</td>
<td>72300.000</td>
<td>0.333333</td>
</tr>
<tr>
<td>200</td>
<td>Clark</td>
<td>45000.000</td>
<td>0.666667</td>
</tr>
<tr>
<td>200</td>
<td>Overbey</td>
<td>39300.000</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

Usage

Sybase IQ calculates the cumulative distribution of a value of x in a set S of size N using:

\[
\text{CUME_DIST}(x) = \frac{\text{number of values in } S \text{ coming before and including } x \text{ in the specified order}}{N}
\]

Composite sort-keys are not currently allowed in the CUME_DIST function. You can use composite sort-keys with any of the other rank functions.
Alphabetical list of functions

You can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. The window-spec must contain the ORDER BY clause, and cannot contain a ROWS or RANGE clause. For information on how to specify the window, see “Analytical functions” on page 104.

**Note** DISTINCT is not supported.

### Standards and compatibility

- **SQL2008** SQL/OLAP feature T612
- **Sybase** Compatible with SQL Anywhere

### DATALENGTH function [System]

**Function**

Returns the length of the expression in bytes.

**Syntax**

```
DATALENGTH ( expression )
```

**Parameters**

- **expression** The expression is usually a column name. If the expression is a string constant, it must be enclosed in quotes.

**Usage**

Table 4-19 lists the return values of DATALENGTH.

#### Table 4-19: DATALENGTH return values

<table>
<thead>
<tr>
<th>Data type</th>
<th>DATALENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>8</td>
</tr>
<tr>
<td>CHAR</td>
<td>Length of the data</td>
</tr>
<tr>
<td>BINARY</td>
<td>Length of the data</td>
</tr>
</tbody>
</table>

**Example**

The following statement returns the value 35, the longest string in the company_name column:

```
SELECT MAX( DATALENGTH( company_name ) )
FROM Customers
```

**Standards and compatibility**

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

**See also**

- “CHAR_LENGTH function [String]” on page 136
- “COL_LENGTH function [System]” on page 138
DATE function [Date and time]

Function
Converts the expression into a date, and removes any hours, minutes, or seconds.

Syntax
```
DATE ( expression )
```

Parameters
- **expression**: The value to be converted to date format. The expression is usually a string.

Example
The following statement returns the value 1988-11-26 as a date.
```
SELECT DATE( '1988-11-26 21:20:53' ) FROM iq_dummy
```

Standards and compatibility
- **SQL92**: Vendor extension
- **Sybase**: Not supported by Adaptive Server Enterprise

DATEADD function [Date and time]

Function
Returns the date produced by adding the specified number of the specified date parts to a date.

Syntax
```
DATEADD ( date-part, numeric-expression, date-expression )
```

Parameters
- **date part**: The date part to be added to the date.
- **numeric-expression**: The number of date parts to be added to the date. The numeric-expression can be any numeric type; the value is truncated to an integer.
- **date-expression**: The date to be modified.

Example
The following statement returns the value 1995-11-02 00:00:00.000:
```
SELECT DATEADD( month, 102, '1987/05/02' ) FROM iq_dummy
```

Usage
DATEADD is a Transact-SQL compatible data manipulation function.

Standards and compatibility
- **SQL92**: Transact-SQL extension.
- **Sybase**: Compatible with Adaptive Server Enterprise.
DATECEILING function [Date and time]

Function

Calculates a new date, time, or datetime value by increasing the provided value up to the nearest larger value of the specified multiple with the specified granularity.

Syntax

DATECEILING (date-part, datetime-expression [.multiple-expression])

Parameters

date part

The date part to be added to the date.

The following date parts are not compatible with DATECEILING:

- DayOfYear
- WeekDay
- CalYearOfWeek
- CalWeekOfYear
- CalDayOfWeek

For a complete listing of date parts, see Date part values on page 113.

datetime-expression

The date, time, or date-time expression containing the value you are evaluating.

multiple-expression (Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by the date_part parameter to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 10-minute intervals. Note that if multiple-expression evaluates to zero, evaluates to a negative number, or is an explicit NULL constant, Sybase IQ generates an error. If this multiple-expression evaluates to a NULL, then the function result is NULL.

Example

This returns the value August 13, 2009 10:40.00.000AM:

```
SELECT DATECEILING( MI, 'August 13, 2009, 10:32.00.132AM', 10) FROM iq_dummy
```

Usage

This function calculates a new date, time, or datetime value by increasing the provided value up to the nearest larger value with the specified granularity. If you include the optional multiple-expression parameter, then the function increases the date and time up to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the multiple-expression parameter.
If you specify a `multiple-expression` for the millisecond, second, minute, or hour date parts, IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, IQ applies two-minute intervals starting at the current hour.

For the millisecond, second, minute, and hour date parts, specify a `multiple-expression` value that divides evenly into the range of the specified date part:

- For milliseconds, the valid `multiple-expression` values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For seconds and minutes, the valid `multiple-expression` values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- For hours, the valid `multiple-expression` values are: 1, 2, 3, 4, 6, 8, 12, 24

If you specify a `multiple-expression` for the day, week, month, quarter, or year date parts, IQ assumes the intervals started at the smallest date value (0000-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0000-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then Sybase IQ calculates 10-day intervals starting at 0000-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a `multiple` that divides evenly into the next larger unit of time granularity.

If IQ rounds to a multiple of the week date part, the date value is always Sunday.

### Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere

---

**DATEDIFF function [Date and time]**

**Function**

Returns the interval between two dates.

**Syntax**

\[
\text{DATEDIFF} ( \text{date-part, date-expression1, date-expression2} )
\]

**Parameters**

- **date-part** Specifies the date part in which the interval is to be measured.
Alphabetical list of functions

For a complete listing of allowed date parts, see “Date parts” on page 112.

date-expression1  The starting date for the interval. This value is subtracted from date-expression2 to return the number of date parts between the two arguments.

date-expression2  The ending date for the interval. date-expression1 is subtracted from this value to return the number of date parts between the two arguments.

Examples

The following statement returns 1:

```
SELECT DATEDIFF( hour, '4:00AM', '5:50AM' ) FROM iq_dummy
```

The following statement returns 102:

```
SELECT DATEDIFF( month, '1987/05/02', '1995/11/15' ) FROM iq_dummy
```

The following statement returns 0:

```
SELECT DATEDIFF( day, '00:00', '23:59' ) FROM iq_dummy
```

The following statement returns 4:

```
SELECT DATEDIFF( day, '1999/07/19 00:00', '1999/07/23 23:59' ) FROM iq_dummy
```

The following statement returns 0:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/07/23' ) FROM iq_dummy
```

The following statement returns 1:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/08/23' ) FROM iq_dummy
```

Usage

This function calculates the number of date parts between two specified dates. The result is a signed integer value equal to (date2 - date1), in date parts.

DATEDIFF results are truncated, not rounded, when the result is not an even multiple of the date part.

When you use day as the date part, DATEDIFF returns the number of midnights between the two times specified, including the second date, but not the first. For example, the following statement returns the value 5. Midnight of the first day 2003/08/03 is not included in the result. Midnight of the second day is included, even though the time specified is before midnight.

```
SELECT DATEDIFF( day, '2003/08/03 14:00', '2003/08/08 14:00' ) FROM iq_dummy
```
When you use month as the date part, DATEDIFF returns the number of first-of-the-months between two dates, including the second date but not the first. For example, both of the following statements return the value 9:

```sql
SELECT DATEDIFF( month, '2003/02/01', '2003/11/15' )
FROM iq_dummy;
SELECT DATEDIFF( month, '2003/02/01', '2003/11/01' )
FROM iq_dummy;
```

The first date 2003/02/01 is a first-of-month, but is not included in the result of either query. The second date 2003/11/01 in the second query is also a first-of-month and is included in the result.

When you use week as the date part, DATEDIFF returns the number of Sundays between the two dates, including the second date but not the first. For example, in the month 2003/08, the dates of the Sundays are 03, 10, 17, 24, and 31. The following query returns the value 4:

```sql
SELECT DATEDIFF( week, '2003/08/03', '2003/08/31' )
FROM iq_dummy;
```

The first Sunday (2003/08/03) is not included in the result.

For smaller time units, there are overflow values:

- **milliseconds** 24 days.
- **seconds** 68 years.
- **minutes** 4083 years.
- **others** No overflow limit

The function returns an overflow error if you exceed these limits.

### Standards and compatibility

- **SQL92** Transact-SQL extension.
- **Sybase** Compatible with Adaptive Server Enterprise.

### DATEFLOOR function [Date and time]

**Function**

Calculates a new date, time, or datetime value by reducing the provided value down to the nearest lower value of the specified multiple with the specified granularity.

**Syntax**

```
DATEDIFF ( date-part, datetime-expression [ , multiple-expression ] )
```

**Parameters**

- **date part** The date part to be added to the date.
The following date parts are not compatible with DATEFLOOR:

- DayofYear
- WeekDay
- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

For a complete listing of date parts, see Date part values on page 113.

**datetime-expression**  The date, time, or date-time expression containing the value you are evaluating.

**multiple-expression**  (Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by the date_part parameter to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 10-minute intervals. Note that if multiple-expression evaluates to zero, evaluates to a negative number, or is an explicit NULL constant, then IQ generates an error. If multiple-expression evaluates to a NULL, then the function result is NULL.

**Example**

The following statement returns the value August 13, 2009 10:35:00.000AM:

```sql
SELECT DATEFLOOR( MINUTE, 'August 13, 2009 10:35.22.123AM') FROM iq_dummy
```

**Usage**

This function calculates a new date, time, or datetime value by reducing the provided value down to the nearest lower value with the specified granularity. If you include the optional multiple-expression parameter, then the function reduces the date and time down to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the multiple-expression parameter.

If you specify a multiple-expression for the millisecond, second, minute, or hour date parts, IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day
For example, if you specify a multiple of two minutes, IQ applies two minute intervals starting at the current hour.

For the millisecond, second, minute, and hour date parts, specify a multiple-expression value that divides evenly into the range of the specified date part:

- For milliseconds, the valid multiple-expression values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For seconds and minutes, the valid multiple-expression values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- For hours, the valid multiple-expression values are: 1, 2, 3, 4, 6, 8, 12, 24

If you specify a multiple-expression for the day, week, month, quarter, or year date parts, IQ assumes the intervals started at the smallest date value (0000-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0000-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then Sybase IQ calculates 10-day intervals starting at 0000-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If IQ rounds to a multiple of the week date part, the date value is always Sunday.

Standards and compatibility

- SQL92 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise or SQL Anywhere

**DATEFORMAT function [Date and time]**

Function Returns a string representing a date expression in the specified format.

Syntax

```
DATEFORMAT ( datetime-expression, string-expression )
```

Parameters

- `datetime-expression` The date/time to be converted. Must be a date, time, timestamp, or character string.
- `string-expression` The format of the converted date.

Example

The following statement returns string values like “Jan 01, 1989”:

```
SELECT DATEFORMAT( start_date, 'Mmm dd, yyyy' ) from Employees;
```

The following statement returns the string “Feb 19, 1987”:

```
SELECT DATEFORMAT( CAST ( '1987/02/19' AS DATE ), 'Mmm Dd, yyyy' ) FROM iq_dummy
```
**DATENAME function [Date and time]**

**Function**

Returns the name of the specified part (such as the month “June”) of a date/time value, as a character string.

**Syntax**

```
DATENAME ( date-part, date-expression )
```

**Parameters**

- `date-part` The date part to be named.
- `date-expression` The date for which the date part name is to be returned. The date must contain the requested `date-part`.

**Example**

The following statement returns the value May:

```
SELECT datename( month , '1987/05/02' ) FROM iq_dummy
```

**Usage**

`DATENAME` returns a character string, even if the result is numeric, such as 23, for the day.

**Standards and compatibility**

- **SQL92** Transact-SQL extension.
• **Sybase** Compatible with Adaptive Server Enterprise.

**DATEPART function [Date and time]**

**Function**
Returns an integer value for the specified part of a date/time value.

**Syntax**

```sql
DATEPART (date-part, date-expression)
```

**Parameters**
- `date-part` The date part to be returned.
- `date-expression` The date for which the part is to be returned. The date must contain the `date-part` field.

**Example**

The following statement returns the value 5:

```sql
SELECT DATEPART ( month, '1987/05/02' ) FROM iq_dummy
```

**Usage**
Note that the `DATE`, `TIME`, and `DTTM` indexes do not support some date parts (Calyearofweek, Calweekofyear, Caldayofweek, Dayofyear, Millisecond).

**Standards and compatibility**
- **SQL92** Transact-SQL extension.
- **Sybase** Compatible with Adaptive Server Enterprise.

**DATEROUND function [Date and time]**

**Function**
Calculates a new date, time, or datetime value by rounding the provided value up or down to the nearest multiple of the specified value with the specified granularity.

**Syntax**

```sql
DATEROUND (date-part, datetime-expression [,multiple-expression])
```

**Parameters**
- `date part` The date part to be added to the date.

The following date parts are not compatible with DATEROUND:

- DayofYear
- WeekDay
- CalYearofWeek
- CalWeekofYear
- CalDayofWeek

For a complete listing of date parts, see Date part values on page 113.
**datetime-expression**  The date, time, or date-time expression containing the value you are evaluating.

**multiple-expression**  (Optional). A nonzero positive integer value expression specifying how many multiples of the units specified by the date_part parameter to use within the calculation. For example, you can use multiple-expression to specify that you want to regularize your data to 10-minute intervals. Note that if multiple-expression evaluates to zero, evaluates to a negative number, or is an explicit NULL constant, then IQ generates an error. If multiple-expression evaluates to a NULL, then the function result is NULL.

**Example**

The following statement returns the value August 13, 2009, 10:30.000AM:

```
SELECT DATEROUND( MI, 'August 13, 2009 10:33.123AM', 10)
FROM iq_dummy
```

**Usage**

This function calculates a new date, time, or datetime value by rounding the provided value up or down to the nearest value with the specified granularity. If you include the optional multiple-expression parameter, then the function rounds the date and time to the nearest specified multiple of the specified granularity.

The data type of the calculated date and time matches the data type of the multiple-expression parameter.

If you specify a multiple-expression for the millisecond, second, minute, or hour date parts, IQ assumes that the multiple applies from the start of the next larger unit of granularity:

- Multiples of millisecond start from the current second
- Multiples of second start from the current minute
- Multiples of minute start from the current hour
- Multiples of hour start from the current day

For example, if you specify a multiple of two minutes, IQ applies two minute intervals starting at the current hour.

For the millisecond, second, minute, and hour date parts, specify a multiple-expression value that divides evenly into the range of the specified date part:

- For milliseconds, the valid multiple-expression values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000
- For seconds and minutes, the valid multiple-expression values are: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
• For hours, the valid multiple-expression values are: 1, 2, 3, 4, 6, 8, 12, 24

If you specify a multiple-expression for the day, week, month, quarter, or year date parts, IQ assumes the intervals started at the smallest date value (0000-01-01), smallest time value (00:00:00.000000), or smallest date-time value (0000-01-01.00:00:00.000000). For example, if you specify a multiple of 10 days, then Sybase IQ calculates 10-day intervals starting at 0000-01-01.

For the day, week, month, quarter, or year date parts, you need not specify a multiple that divides evenly into the next larger unit of time granularity.

If IQ rounds to a multiple of the week date part, then the date value is always Sunday.

### Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere

---

**DATETIME function [Date and time]**

**Function**
Converts an expression into a timestamp.

**Syntax**
```
DATETIME ( expression )
```

**Parameters**
- **expression** The expression to be converted. The expression is usually a string. Conversion errors may be reported.

**Example**
The following statement returns a timestamp with value 1998-09-09 12:12:12.000:

```
SELECT DATETIME( '1998-09-09 12:12:12.000' ) FROM iq_dummy
```

**Standards and compatibility**
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

---

**DAY function [Date and time]**

**Function**
Returns an integer from 1 to 31 corresponding to the day of the month of the date specified.

**Syntax**
```
DAY ( date-expression )
```

**Parameters**
- **date-expression** The date.

**Example**
The following statement returns the value 12:
Alphabetical list of functions

SELECT DAY ( '2001-09-12' ) FROM iq_dummy

Dayname function [Date and time]

Function
Returns the name of the day of the week from the specified date.

Syntax
DAYNAME ( date-expression )

Parameters
date-expression The date.

Example
The following statement returns the value Saturday:

SELECT DAYNAME ( '1987/05/02' ) FROM iq_dummy

Standards and compatibility
• SQL92 Vendor extension
• Sybase Compatible with Adaptive Server Enterprise.

Days function [Date and time]

Function
Returns the number of days since an arbitrary starting date, returns the number of days between two specified dates, or adds the specified integer-expression number of days to a given date.

Syntax
DAYS ( datetime-expression )
| ( datetime-expression, datetime-expression )
| ( datetime-expression, integer-expression )

Parameters
datetime-expression A date and time.

integer-expression The number of days to be added to the datetime-expression. If the integer-expression is negative, the appropriate number of days are subtracted from the date/time. If you supply an integer expression, the datetime-expression must be explicitly cast as a date.

For information on casting data types, see “CAST function [Data type conversion]” on page 133.

DAYS ignores hours, minutes, and seconds.

Examples
The following statement returns the integer value 729948:

SELECT DAYS ( '1998-07-13 06:07:12' ) FROM iq_dummy
The following statement returns the integer value -366, which is the difference between the two dates:

```
SELECT DAYS('1998-07-13 06:07:12',
'1997-07-12 10:07:12') FROM iq_dummy
```

The following statement returns the value 1999-07-14:

```
SELECT DAYS(CAST('1998-07-13' AS DATE), 366)
FROM iq_dummy
```

### Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  Not supported by Adaptive Server Enterprise

### DB_ID function [System]

**Function**
Returns the database ID number.

**Syntax**

```
DB_ID([database-name])
```

**Parameters**

database-name  A string expression containing the database name. If
 database-name is a string constant, it must be enclosed in quotes. If no
database-name is supplied, the ID number of the current database is returned.

**Examples**

The following statement returns the value 0, if iqdemo is the only running database:

```
SELECT DB_ID('iqdemo') FROM iq_dummy
```

The following statement returns the value 0, if executed against the only running database:

```
SELECT DB_ID() FROM iq_dummy
```

**Standards and compatibility**

- **SQL92**  Vendor extension
- **Sybase**  Adaptive Server Enterprise function implemented for Sybase IQ

**See also**

“DB_NAME function [System]” on page 162
“OBJECT_ID function [System]” on page 215
DB_NAME function [System]

Function
Returns the database name.
Syntax
\[
\text{DB_NAME} ( [ \text{database-id} ] )
\]

Note

Parameters
- database-id The ID of the database. database-id must be a numeric expression.

Example
The following statement returns the database name iqdemo, when executed against the sample database:

\[
\text{SELECT DB_NAME( 0 ) FROM iq_dummy}
\]

Usage
If no database-id is supplied, the name of the current database is returned.

Standards and compatibility
- SQL92 Vendor extension
- Sybase Adaptive Server Enterprise function implemented for Sybase IQ

See also
“COL_NAME function [System]” on page 138
“DB_ID function [System]” on page 161
“OBJECT_NAME function [System]” on page 216

DB_PROPERTY function [System]

Function
Returns the value of the given property.
Syntax
\[
\text{DB_PROPERTY} ( \{ \text{property-id} | \text{property-name} \} , \{ \text{database-id} | \text{database-name} \} )
\]

Note

Parameters
- property-id The database property ID.
- property-name The database property name.
- database-id The database ID number, as returned by DB_ID. Typically, the database name is used.
database-name  The name of the database, as returned by DB_NAME.

Example

The following statement returns the page size of the current database, in bytes.

```
SELECT DB_PROPERTY( 'PAGESIZE' ) FROM iq_dummy
```

Usage

Returns a string. The current database is used if the second argument is omitted.

Standards and compatibility

- SQL2  Vendor extension
- Sybase  Not supported by Adaptive Server Enterprise

See also

- “Properties available for each database” on page 122
- “DB_ID function [System]” on page 161
- “DB_NAME function [System]” on page 162

DEGREES function [Numeric]

Function

Converts a number from radians to degrees.

Syntax

```
DEGREES ( numeric-expression )
```

Parameters

numeric-expression  An angle in radians.

Example

The following statement returns the value 29.793805:

```
SELECT DEGREES( 0.52 ) FROM iq_dummy
```

Standards and compatibility

- SQL2  Vendor extension
- Sybase  Compatible with Adaptive Server Enterprise

DENSE_RANK function [Analytical]

Function

Ranks items in a group.

Syntax

```
DENSE_RANK () OVER ( ORDER BY expression [ ASC | DESC ] )
```

Parameters

expression  A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Example

The following statement illustrates the use of the DENSE_RANK function:

```
SELECT s_suppkey, DENSE_RANK() OVER ( ORDER BY SUM(s_acctBal) DESC ) AS rank_dense FROM supplier GROUP BY s_suppkey;
```
Usage

DENSE_RANK is a rank analytical function. The dense rank of row R is defined as the number of rows preceding and including R that are distinct within the groups specified in the OVER clause or distinct over the entire result set. The difference between DENSE_RANK and RANK is that DENSE_RANK leaves no gap in the ranking sequence when there is a tie. RANK leaves a gap when there is a tie.

DENSE_RANK requires an OVER (ORDER BY) clause. The ORDER BY clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the OVER clause and is not an ORDER BY for the SELECT. No aggregation functions in the rank query are allowed to specify DISTINCT.

The OVER clause indicates that the function operates on a query result set. The result set is the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have all been evaluated. The OVER clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

DENSE_RANK is allowed only in the select list of a SELECT or INSERT statement or in the ORDER BY clause of the SELECT statement. DENSE_RANK can be in a view or a union. The DENSE_RANK function cannot be used in a subquery, a HAVING clause, or in the select list of an UPDATE or DELETE statement. Only one rank analytical function is allowed per query.

Standards and compatibility

• SQL92 Vendor extension
• Sybase Not supported by Adaptive Server Enterprise or SQL Anywhere

See also

“Analytical functions” on page 104
“RANK function [Analytical]” on page 229
DIFFERENCE function [String]

Function
Compares two strings, evaluates the similarity between them, and returns a
value from 0 to 4. The best match is 4.

Syntax
DIFFERENCE ( string-expression1, string-expression2 )

Parameters
string-expression1 The first string to compare.
string-expression2 The second string to compare.

Examples
The following statement returns the value 4:
SELECT DIFFERENCE( 'Smith', 'Smith' ) FROM iq_dummy
The following statement returns the value 4:
SELECT DIFFERENCE( 'Smith', 'Smyth' ) FROM iq_dummy
The following statement returns the value 3:
SELECT DIFFERENCE( 'Smith', 'Sweeney' ) FROM iq_dummy
The following statement returns the value 2:
SELECT DIFFERENCE( 'Smith', 'Jones' ) FROM iq_dummy
The following statement returns the value 1:
SELECT DIFFERENCE( 'Smith', 'Rubin' ) FROM iq_dummy
The following statement returns the value 0:
SELECT DIFFERENCE( 'Smith', 'Wilkins' ) FROM iq_dummy

Standards and compatibility
• SQL92 Vendor extension
• Sybase Compatible with Adaptive Server Enterprise.

See also
“SOUNDEX function [String]” on page 255

DOW function [Date and time]

Function
Returns a number from 1 to 7 representing the day of the week of the specified
date, with Sunday=1, Monday=2, and so on.

Syntax
DOW ( date-expression )
Parameters
date-expression The date.

Example
The following statement returns the value 5:
SELECT DOW( '1998-07-09' ) FROM iq_dummy
Usage
See DATE_FIRST_DAY_OF_WEEK option in Reference: Statements and Options if you need Monday (or another day) to be the first day of the week.

Standards and compatibility
- **SQL92**  Vendor extension
- **Sybase**  Not supported by Adaptive Server Enterprise

**ERRORMSG function [Miscellaneous]**

**Function**
Provides the error message for the current error, or for a specified SQLSTATE or SQLCODE value.

**Syntax**
```
ERRORMSG ( [ sqlstate | sqlcode ] )
```

- `sqlstate`: string
- `sqlcode`: integer

**Parameters**
- **sqlstate**  The SQLSTATE value for which the error message is to be returned.
- **sqlcode**  The SQLCODE value for which the error message is to be returned.

**Example**
The following statement returns the error message for SQLCODE -813:
```
select errormsg( -813 )
```

**Return value**
A string containing the error message. If no argument is supplied, the error message for the current state is supplied. Any substitutions (such as table names and column names) are made.

If an argument is supplied, the error message for the supplied SQLSTATE or SQLCODE is returned, with no substitutions. Table names and column names are supplied as placeholders ("???").

The `ERRORMSG` function returns SQL Anywhere and Sybase IQ error messages.

**Standards and compatibility**
- **SQL92**  Vendor extension
- **SQL99**  Vendor extension
- **Sybase**  Not supported by Adaptive Server Enterprise

**See also**
*System Administration Guide: Volume 2*

**EVENT_CONDITION function [System]**

**Function**
To specify when an event handler is triggered.
EVENT_CONDITION ( condition-name )


Parameters

condition-name The condition triggering the event. The possible values are preset in the database, and are case-insensitive. Each condition is valid only for certain event types. Table 4-20 lists the conditions and the events for which they are valid.

Table 4-20: Valid conditions for events

<table>
<thead>
<tr>
<th>Condition name</th>
<th>Units</th>
<th>Valid for</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBFreePercent</td>
<td>N/A</td>
<td>DBDiskSpace</td>
<td>DBDiskSpace shows free space in the system database file (.db file), not the IQ store.</td>
</tr>
<tr>
<td>DBFreeSpace</td>
<td>Megabytes</td>
<td>DBDiskSpace</td>
<td></td>
</tr>
<tr>
<td>DBSize</td>
<td>Megabytes</td>
<td>GrowDB</td>
<td></td>
</tr>
<tr>
<td>ErrorNumber</td>
<td>N/A</td>
<td>RAISERROR</td>
<td></td>
</tr>
<tr>
<td>IdleTime</td>
<td>Seconds</td>
<td>ServerIdle</td>
<td></td>
</tr>
<tr>
<td>Interval</td>
<td>Seconds</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>LogFreePercent</td>
<td>N/A</td>
<td>LogDiskSpace</td>
<td></td>
</tr>
<tr>
<td>LogFreeSpace</td>
<td>Megabytes</td>
<td>LogDiskSpace</td>
<td></td>
</tr>
<tr>
<td>LogSize</td>
<td>Megabytes</td>
<td>GrowLog</td>
<td></td>
</tr>
<tr>
<td>RemainingValues</td>
<td>Integer</td>
<td>GlobalAutoincrement</td>
<td>The number of remaining values.</td>
</tr>
<tr>
<td>TempFreePercent</td>
<td>N/A</td>
<td>TempDiskSpace</td>
<td>TempDiskSpace shows free space in the system temporary file (pointed to by TEMP or IQTMP15 environment variable), not the IQ temporary store.</td>
</tr>
<tr>
<td>TempFreeSpace</td>
<td>Megabytes</td>
<td>TempDiskSpace</td>
<td></td>
</tr>
<tr>
<td>TempSize</td>
<td>Megabytes</td>
<td>GrowTemp</td>
<td></td>
</tr>
</tbody>
</table>

Example

The following event definition uses the EVENT_CONDITION function:
Alphabetical list of functions

create event LogNotifier
type LogDiskSpace
where event_condition( 'LogFreePercent' ) < 50
handler
begin
    message 'LogNotifier message'
end

Standards and compatibility
• SQL92  Vendor extension
• Sybase  Not supported by Adaptive Server Enterprise

See also
CREATE EVENT statement in Reference: Statements and Options

EVENT_CONDITION_NAME function [System]

Function
Can be used to list the possible parameters for EVENT_CONDITION.

Syntax
EVENT_CONDITION_NAME ( integer )

Parameters
integer  Must be greater than or equal to zero.

Usage
You can use EVENT_CONDITION_NAME to obtain a list of all
EVENT_CONDITION arguments by looping over integers until the function
returns NULL.

Standards and compatibility
• SQL92  Vendor extension
• Sybase  Not supported by Adaptive Server Enterprise

See also
CREATE EVENT statement in Reference: Statements and Options

EVENT_PARAMETER function [System]

Function
Provides context information for event handlers.

Syntax
EVENT_PARAMETER ( context-name )
context-name:
    'ConnectionID'
    'User'
    'EventName'

168  Sybase IQ
### Parameters

**context-name**  
One of the preset strings. The strings are case-insensitive, and carry the following information:

- **ConnectionId**  
The connection ID, as returned by  
  
  \[
  \text{connection\_property}\left(\text{'}id\text{'}\right)
  \]

- **User**  
The user ID for the user that caused the event to be triggered.

- **EventManager**  
The name of the event that has been triggered.

- **Executions**  
The number of times the event handler has been executed.

- **NumActive**  
The number of active instances of an event handler. This is useful if you want to limit an event handler so that only one instance executes at any given time.

- **TableName**  
The name of the table, for use with RemainingValues.

In addition, you can access any of the valid *condition-name* arguments to the **EVENT\_CONDITION** function from the **EVENT\_PARAMETER** function.

### Standards and compatibility

- **SQL92**  
Vendor extension

- **Sybase**  
Not supported by Adaptive Server Enterprise

### See also

“**EVENT\_CONDITION** function [System]” on page 166

CREATE EVENT statement in **Reference: Statements and Options**

### EXP function [Numeric]

**Function**  
Returns the exponential function, e to the power of a number.

**Syntax**  

\[
\text{EXP}\left(\text{numeric\_expression}\right)
\]

**Parameters**  

- **numeric-expression**  
The exponent.

**Example**  
The following statement returns the value 3269017.3724721107:
EXP_WEIGHTED_AVG function [Aggregate]

Function
Calculates an exponential weighted moving average. Weightings determine the relative importance of each quantity that makes up the average.

Syntax

```
EXP_WEIGHTED_AVG (expression, period-expression)
OVER (window-spec)
```

```
window-spec: See the Usage section, below.
```

Parameters
- **expression**: A numeric expression for which a weighted value is being computed.
- **period-expression**: A numeric expression specifying the period for which the average is to be computed.

Usage
Similar to the WEIGHTED_AVG function, the weights in EXP_WEIGHTED_AVG decrease over time. However, weights in WEIGHTED_AVG decrease arithmetically, whereas weights in EXP_WEIGHTED_AVG decrease exponentially. Exponential weighting applies more weight to the most recent values, and decreases the weight for older values while still applying some weight.

Sybase IQ calculates the exponential moving average using:

```
S*C+(1-S)*PEMA
```

In the calculation above, IQ applies the smoothing factor by multiplying the current closing price (C) by the smoothing constant (S) added to the product of the previous day’s exponential moving average value (PEMA) and 1 minus the smoothing factor.

Sybase IQ calculates the exponential moving average over the entire period specified by the OVER clause. period-expression specifies the moving range of the exponential moving average.
You can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. The window-spec must contain an ORDER BY statement and cannot contain a frame specification. For information on how to specify the window, see “Analytical functions” on page 104.

**Note** ROLLUP and CUBE are not supported in the GROUP BY clause. DISTINCT is not supported.

Example

The following example returns an exponential weighted average of salaries for employees in Florida with the salary of recently hired employees contributing the most weight to the average. There are three rows used in the weighting:

```sql
SELECT DepartmentID, Surname, Salary,
EXP_WEIGHTED_AVG(Salary, 3) OVER (ORDER BY YEAR(StartDate) DESC) as "W_AVG"
FROM Employees
WHERE State IN ('FL') ORDER BY StartDate DESC
```

The returned result set is:

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>Surname</th>
<th>Salary</th>
<th>W_AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Evans</td>
<td>68940.000</td>
<td>34470.000000</td>
</tr>
<tr>
<td>300</td>
<td>Litton</td>
<td>58930.000</td>
<td>46700.000000</td>
</tr>
<tr>
<td>200</td>
<td>Sterling</td>
<td>64900.000</td>
<td>55800.000000</td>
</tr>
<tr>
<td>200</td>
<td>Kelly</td>
<td>87500.000</td>
<td>71650.000000</td>
</tr>
<tr>
<td>400</td>
<td>Charlton</td>
<td>28300.000</td>
<td>49975.000000</td>
</tr>
<tr>
<td>100</td>
<td>Lull</td>
<td>87900.000</td>
<td>68937.500000</td>
</tr>
<tr>
<td>100</td>
<td>Gowda</td>
<td>59840.000</td>
<td>60621.875000</td>
</tr>
<tr>
<td>400</td>
<td>Francis</td>
<td>53870.000</td>
<td>61403.750000</td>
</tr>
</tbody>
</table>

**Standards and compatibility**
- **SQL2008** Vendor extension

**FIRST_VALUE function [Aggregate]**

Function  
Returns the first value from a set of values.

**Syntax**  
```sql
FIRST_VALUE (expression [IGNORE NULLS | RESPECT NULLS])
```
OVER (window-spec)

Parameters

expression  The expression on which to determine the first value in an ordered set.

Usage

FIRST_VALUE returns the first value in a set of values, which is usually an ordered set. If the first value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. If you specify IGNORE NULLS, then FIRST_VALUE returns the first non-null value in the set, or NULL if all values are null.

The data type of the returned value is the same as that of the input value.

You cannot use FIRST_VALUE or any other analytic function for expression. That is, you cannot nest analytic functions, but you can use other built-in function expressions for expression.

If the window-spec does not contain an ORDER BY, then the result is arbitrary. If there is no window-spec, the answer is arbitrary.

You can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Note  DISTINCT is not supported.

Example

The following example returns the relationship, expressed as a percentage, between each employee’s salary and that of the most recently hired employee in the same department:

```
SELECT DepartmentID, EmployeeID,
100 * Salary / ( FIRST_VALUE( Salary ) OVER ( PARTITION BY DepartmentID ORDER BY Year(StartDate) DESC ) )
AS percentage
FROM Employees order by DepartmentID DESC;
```

The returned result set is:
Table 4-22: FIRST_VALUE result set

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>EmployeeID</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1658</td>
<td>100.00000000000000000000</td>
</tr>
<tr>
<td>500</td>
<td>1570</td>
<td>138.842709713689113761394</td>
</tr>
<tr>
<td>500</td>
<td>1615</td>
<td>110.428462434244870095972</td>
</tr>
<tr>
<td>500</td>
<td>1013</td>
<td>109.58519053929245472330</td>
</tr>
<tr>
<td>500</td>
<td>730</td>
<td>137.734409508894510701521</td>
</tr>
<tr>
<td>500</td>
<td>921</td>
<td>167.44970485483666654619</td>
</tr>
<tr>
<td>500</td>
<td>868</td>
<td>113.239368750752921334778</td>
</tr>
<tr>
<td>500</td>
<td>703</td>
<td>222.867927558928643135365</td>
</tr>
<tr>
<td>500</td>
<td>191</td>
<td>119.66429747419989595908</td>
</tr>
<tr>
<td>400</td>
<td>1684</td>
<td>100.00000000000000000000</td>
</tr>
<tr>
<td>400</td>
<td>1740</td>
<td>76.128652163477274215016</td>
</tr>
<tr>
<td>400</td>
<td>1751</td>
<td>76.353400685155687446813</td>
</tr>
<tr>
<td>400</td>
<td>1607</td>
<td>133.758100765890593292456</td>
</tr>
<tr>
<td>400</td>
<td>1507</td>
<td>77.996465120338650199655</td>
</tr>
<tr>
<td>400</td>
<td>1576</td>
<td>150.428767810774836893669</td>
</tr>
</tbody>
</table>

In this example, employee 1658 is the first row for department 500, indicating that employee 1658 is the most recent hire in that department, and therefore receives a percentage of 100%. Percentages for the remaining employees in department 500 are calculated relative to that of employee 1658. For example, employee 1570 earns approximately 139% of what employee 1658 earns.

**Stands and compatibility**
- **SQL2008** SQL/OLAP feature T612
- **Sybase** Compatible with SQL Anywhere

### FLOOR function [Numeric]

**Function**
Returns the floor of (largest integer not greater than) a number.

**Syntax**

FLOOR ( numeric-expression )

**Parameters**

- **numeric-expression** The number, usually a float.

**Examples**

The following statement returns the value 123.00:

```sql
SELECT FLOOR ( 123.00 ) FROM iq_dummy
```

The following statement returns the value 123:

```sql
SELECT FLOOR ( 123.45 ) FROM iq_dummy
```
Alphabetical list of functions

The following statement returns the value -124.00.

```
SELECT FLOOR ( -123.45 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  Compatible with Adaptive Server Enterprise

See also

“CEILING function [Numeric]” on page 135

GETDATE function [Date and time]

Function

Returns the current date and time.

Syntax

```
GETDATE()
```

Example

The following statement returns the system date and time.

```
SELECT GETDATE() FROM iq_dummy
```

Usage

GETDATE is a Transact-SQL compatible data manipulation function.

Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  Compatible with Adaptive Server Enterprise

GRAPHICAL_PLAN function [String]

Function

Returns the graphical query plan to Interactive SQL in an XML format string.

Syntax

```
GRAPHICAL_PLAN( string-expression )
```


Parameters

- **string-expression**  SQL statement for which the plan is to be generated.

  *string-expression* is generally a SELECT statement, but it can also be an UPDATE or DELETE, INSERT SELECT, or SELECT INTO statement.

  If you do not provide an argument to the GRAPHICAL_PLAN function, the query plan is returned to you from the cache. If there is no query plan in the cache, then this message appears:

```
plan not available
```
The behavior of GRAPHICAL_PLAN function is controlled by database options QUERY_PLAN_TEXT_ACCESS and QUERY_PLAN_TEXT_CACHING. If QUERY_PLAN_TEXT_ACCESS is OFF (the default), then this message appears:

Plan not available. The database option QUERY_PLAN_TEXT_ACCESS is OFF

If a user needs access to the plan, the DBA must set option QUERY_PLAN_TEXT_ACCESS ON for that user.

If QUERY_PLAN_TEXT_ACCESS is ON, and the query plan for the string expression is available in the cache maintained on the server, the query plan from the cache is returned to you.

If the query plan is not available in the cache and you are authorized to view plans on the client, then a query plan with optimizer estimates (query plan with NOEXEC option ON) is generated and appears on the dbisql client plan window.

---

**Note** Sybase IQ does not support NOEXEC plan generation for SELECT, UPDATE, DELETE, INSERT SELECT, and SELECT INTO queries.

---

When a user requests a query plan that has not yet been executed, the query plan is not available in the cache. Instead, a query plan with optimizer estimates is returned without QUERY_PLAN_AFTER_RUN statistics.

Query plans for stored procedures are not accessible using the GRAPHICAL_PLAN function.

Users can view the query plan for cursors opened for IQ queries. A cursor is declared and opened using DECLARE CURSOR and OPEN CURSOR commands. To obtain the query plan for the most recently opened cursor, use:

```sql
SELECT GRAPHICAL_PLAN();
```

With the QUERY_PLAN_AFTER_RUN option OFF, the plan appears after OPEN CURSOR or CLOSE CURSOR. However, if QUERY_PLAN_AFTER_RUN is ON, CLOSE CURSOR must be executed before you request the plan.
For information on viewing the query optimizer's execution plan for a SQL statement in the Plan Viewer window in Interactive SQL, see “Viewing graphical plans in Interactive SQL” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Administering Your Database > SQL Anywhere graphical administration tools > Using Interactive SQL > Viewing plans using the Interactive SQL Plan Viewer.

When Interactive SQL users request plans for UPDATE, DELETE, SELECT INTO, and INSERT SELECT queries, the NOEXEC plan is not supported. To access the query plan, first explicitly execute the query, and then request the plan.

The following example passes a SELECT statement as a string parameter and returns the plan for executing the query. It saves the plan in the file gplan.xml.

```
Note If you use the OUTPUT statement’s HEXADECIMAL clause set to ASIS to get formatted plan output, the values of characters are written without any escaping, even if the value contains control characters. ASIS is useful for text that contains formatting characters such as tabs or carriage returns.

SELECT GRAPHICAL_PLAN ('SELECT * FROM Employees') ; OUTPUT to 'C:\gplan.xml' HEXADECIMAL ASIS quote '';

The following example returns the query plan from the cache, if available:

SELECT GRAPHICAL_PLAN ( ) ;
```

### Standards and compatibility

- **SQL92** Vendor extension
- **SQL99** SQL/foundation feature outside of core SQL
- **Sybase** Not supported by Adaptive Server Enterprise

### See also

“HTML_PLAN function [String]” on page 183

“OUTPUT statement [DBISQL]” in Reference: Building Blocks, Tables, and Procedures

“NOEXEC option,” “QUERY_PLAN_AFTER_RUN option,”
“QUERY_PLAN_AS_HTML option,” “QUERY_PLAN_TEXT_ACCESS option,” and “QUERY_PLAN_TEXT_CACHING option” in Reference: Statements and Options
GROUPING function [Aggregate]

Function

Identifies whether a column in a ROLLUP or CUBE operation result set is NULL because it is part of a subtotal row, or NULL because of the underlying data.

Syntax

GROUPING ( group-by-expression )

Parameters

group-by-expression  An expression appearing as a grouping column in the result set of a query that uses a GROUP BY clause with the ROLLUP or CUBE keyword. The function identifies subtotal rows added to the result set by a ROLLUP or CUBE operation.

Currently, Sybase IQ does not support the PERCENTILE_CONT or PERCENTILE_DISC functions with GROUP BY CUBE operations.

Return value

• 1 Indicates that group-by-expression is NULL because it is part of a subtotal row. The column is not a prefix column for that row.

• 0 Indicates that group-by-expression is a prefix column of a subtotal row.

Standards and compatibility

• SQL92  Vendor extension

• SQL99  SQL/foundation feature outside of core SQL

• Sybase  Not supported by Adaptive Server Enterprise

See also

SELECT statement in Reference: Statements and Options


GROUP_MEMBER function [System]

Function

Identifies whether the user belongs to the specified group.

Syntax

GROUP_MEMBER ( group-name-string-expression[ , user-name-string-expression ] )

Parameters

group-name-string-expression  Identifies the group to be considered.

user-name-string-expression  Identifies the user to be considered. If not supplied, then the current user name is assumed.
Return value

- **0**: Returns 0 if the group does not exist, if the user does not exist, or if the user does not belong to the specified group.
- **1**: Returns an integer other than 0 if the user is a member of the specified group.

Standards and compatibility

- **SQL92**: Vendor extension
- **SQL99**: SQL/foundation feature outside of core SQL
- **Sybase**: Not supported by Adaptive Server Enterprise

**HEXTOBIGINT function [Data type conversion]**

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns the BIGINT equivalent of a hexadecimal string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>HEXTOBIGINT ( hexadecimal-string )</code></td>
</tr>
<tr>
<td>Parameters</td>
<td><strong>hexadecimal-string</strong>: The hexadecimal value to be converted to a big integer (BIGINT). Input can be in the following forms, with either a lowercase or uppercase “0x” in the prefix, or no prefix:</td>
</tr>
<tr>
<td>Example</td>
<td>The following statements return the value 4294967287:</td>
</tr>
<tr>
<td>Usage</td>
<td>The HEXTOBIGINT function accepts hexadecimal integers and returns the BIGINT equivalent. Hexadecimal integers can be provided as CHAR and VARCHAR value expressions, as well as BINARY and VARBINARY expressions. The HEXTOBIGINT function accepts a valid hexadecimal string, with or without a “0x” or “0X” prefix, enclosed in single quotes. Input of fewer than 16 digits is assumed to be left-padded with zeros. For data type conversion failure on input, Sybase IQ returns an error unless the CONVERSION_ERROR option is set to OFF. When CONVERSION_ERROR is OFF, invalid hexadecimal input returns NULL. An error is returned if a BINARY or VARBINARY value exceeds 8 bytes and a CHAR or VARCHAR value exceeds 16 characters, with the exception of the value being appended with ‘0x.’</td>
</tr>
</tbody>
</table>
HEXTOINT function [Data type conversion]

Function
Returns the unsigned BIGINT equivalent of a hexadecimal string.

Syntax
HEXTOINT (hexadecimal-string)

Parameters
hexadecimal-string The string to be converted to an integer. Input can be in the following forms, with either a lowercase or uppercase “x” in the prefix, or no prefix:

0xhex-string
0Xhex-string
hex-string

Examples
The following statements return the value 420:

SELECT HEXTOINT ('0x1A4') FROM iq_dummy
SELECT HEXTOINT ('0X1A4') FROM iq_dummy
SELECT HEXTOINT ('1A4') FROM iq_dummy

Usage
For invalid hexadecimal input, Sybase IQ returns an error unless the CONVERSION_ERROR option is OFF. When CONVERSION_ERROR is OFF, invalid hexadecimal input returns NULL.

The database option ASE_FUNCTION_BEHAVIOR specifies that output of Sybase IQ functions, including INTTOHEX and HEXTOINT, is consistent with the output of Adaptive Server Enterprise functions. When the ASE_FUNCTION_BEHAVIOR option is ON:

• Sybase IQ HEXTOINT assumes input is a hexadecimal string of 8 characters; if the length is less than 8 characters long, the string is left padded with zeros.

• Sybase IQ HEXTOINT accepts a maximum of 16 characters prefixed with 0x (a total of 18 characters); use caution, as a large input value can result in an integer value that overflows the 32-bit signed integer output size.
Alphabetical list of functions

• The data type of the output of the Sybase IQ HEXTOINT function is assumed to be a 32-bit signed integer.

• Sybase IQ HEXTOINT accepts a 32-bit hexadecimal integer as a signed representation.

• For more than 8 hexadecimal characters, Sybase IQ HEXTOINT considers only relevant characters.

Standards and compatibility

• SQL92 Vendor extension

• Sybase Compatible with Adaptive Server Enterprise

See also

ASE_FUNCTION_BEHAVIOR option in Reference: Statements and Options

CONVERSION_ERROR option [TSQL] in Reference: Statements and Options

“INTTOHEX function [Data type conversion]” on page 189

HOUR function [Date and time]

Function

Returns a number from 0 to 23 corresponding to the hour component of the specified date/time.

Syntax

HOUR ( datetime-expression )

Parameters

datetime-expression The date/time.

Example

The following statement returns the value 21:

SELECT HOUR( '1998-07-09 21:12:13' ) FROM iq_dummy

Standards and compatibility

• SQL92 Vendor extension

• Sybase Not supported by Adaptive Server Enterprise

HOURS function [Date and time]

Function

Returns the number of hours since an arbitrary starting date and time, the number of whole hours between two specified times, or adds the specified integer-expression number of hours to a time.

Syntax

HOURS ( datetime-expression )

| date-time-expression, date-time-expression |
| date-time-expression, integer-expression |

Parameters

datetime-expression A date and time.
integer-expression  The number of hours to be added to the datetime-expression. If integer-expression is negative, the appropriate number of hours are subtracted from the date/time. If you supply an integer expression, the datetime-expression must be explicitly cast as a datetime data type.

For information on casting data types, see “CAST function [Data type conversion]” on page 133.

Examples

The following statement returns the value 17518758:

```
SELECT HOURS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 4, to signify the difference between the two times:

```
SELECT HOURS( '1999-07-13 06:07:12',
             '1999-07-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-05-13 02:05:07.000:

```
SELECT HOURS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 ) FROM iq_dummy
```

Usage

The second syntax returns the number of whole hours from the first date/time to the second date/time. The number might be negative.

Standards and compatibility

- SQL92  Vendor extension
- Sybase  Not supported by Adaptive Server Enterprise

**HTML_DECODE function [HTTP]**

Function

Decodes special character entities that appear in HTML literal strings.

Syntax

```
HTML_DECODE ( string )
```


Parameters

- **string**  An arbitrary literal string used in an HTML document.

Usage

This function returns the string argument after making the following set of substitutions:

<table>
<thead>
<tr>
<th>Characters</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>&quot;</td>
</tr>
</tbody>
</table>
### HTML_ENCODE function [HTTP]

**Function**

Encodes special characters within strings to be inserted into HTML documents.

**Syntax**

```
HTML_ENCODE ( string )
```

**Parameters**

- `string` An arbitrary literal string used in an HTML document.

**Usage**

This function returns the string argument after making the following set of substitutions:

<table>
<thead>
<tr>
<th>Characters</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>codes no less than 0X20</td>
<td>&amp;#xnn</td>
</tr>
</tbody>
</table>

**Notes**


### Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

- “HTML_ENCODE function [HTTP]” on page 182
- “HTTP_ENCODE function [HTTP]” on page 185

---

**Alphabetical list of functions**

<table>
<thead>
<tr>
<th>Characters</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>codes no less than 0X20</td>
<td>&amp;#xnn</td>
</tr>
</tbody>
</table>
HTML_PLAN function [String]

Function

Returns query plans in an HTML format string.

Syntax

HTML_PLAN ( string-expression )

Parameters

string-expression  SQL statement for which the plan is to be generated. It is primarily a SELECT statement but can be an UPDATE or DELETE statement.

If you do not provide an argument to the HTML_PLAN function, the query plan is returned to you from the cache. If there is no query plan in the cache, this message appears:

No plan available

The behavior of the HTML_PLAN function is controlled by database options QUERY_PLAN_TEXT_ACCESS and QUERY_PLAN_TEXT_CACHING. If QUERY_PLAN_TEXT_ACCESS is OFF (the default), this message appears:

Plan not available. The database option QUERY_PLAN_TEXT_ACCESS is OFF

If QUERY_PLAN_TEXT_ACCESS is ON, and the query plan for the string expression is available in the cache maintained on the server, the query plan from the cache is returned to you.

The HTML_PLAN function can be used to return query plans to Interactive SQL using SELECT, UPDATE, DELETE, INSERT SELECT, and SELECT INTO.

Users can view the query plan for cursors opened for IQ queries. To obtain the query plan for the most recently opened cursor, use:

SELECT HTML_PLAN ( )

With QUERY_PLAN_AFTER_RUN option OFF, the plan appears after OPEN CURSOR or CLOSE CURSOR. However, if QUERY_PLAN_AFTER_RUN is ON, CLOSE CURSOR must be executed before you request the plan.
For information on viewing the query optimizer's execution plan for a SQL statement in the Plan Viewer window in Interactive SQL, see “Viewing plans using the Interactive SQL Plan Viewer " in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Administering Your Database > SQL Anywhere graphical administration tools > Using Interactive SQL.

When Interactive SQL users request plans for UPDATE, DELETE, SELECT INTO, and INSERT SELECT queries, the NOEXEC plan is not supported. To access the query plan, first explicitly execute the query, then request the plan.

When you request an HTML_PLAN for a SQL Anywhere query or for an OMNI/CIS decomposed query, the following message is returned:

No plan. HTML_PLAN function is not supported for this type of statement or database.

Examples

The following example passes a SELECT statement as a string parameter and returns the HTML plan for executing the query. It saves the plan in the file hplan.html.

```
SELECT HTML_PLAN ('SELECT * FROM Employees');
OUTPUT to 'C:\hplan.html' HEXADECIMAL ASIS QUOTE '';
```

The OUTPUT TO clause HEXADECIMAL ASIS is useful for text that contains formatting characters such as tabs or carriage returns. When set to ASIS, values are written as is, without any escaping, even if the values contain control characters.

The following example returns the HTML query plan from the cache, if available.

```
SELECT HTML_PLAN ( );
```

Standards and compatibility

• **SQL92**  Vendor extension

• **SQL99**  SQL/foundation feature outside of core SQL

• **Sybase**  Not supported by Adaptive Server Enterprise

See also

“GRAPHICAL_PLAN function [String]” on page 174

“OUTPUT statement [DBISQL]” in Reference: Building Blocks, Tables, and Procedures

“NOEXEC option,” “QUERY_PLAN_AFTER_RUN option,”
“QUERY_PLAN_AS_HTML option,” “QUERY_PLAN_TEXT_ACCESS option," and “QUERY_PLAN_TEXT_CACHING option” in Reference: Statements and Options
HTTP_DECODE function [HTTP]

Function
Decodes special characters within strings for use with HTTP.

Syntax
```
HTTP_DECODE ( string )
```

Note

Parameters
- **string** Arbitrary string to be used in an HTTP request.

Usage
This function returns the string argument after replacing all character sequences of the form %nn, where nn is a hexadecimal value, with the character with code nn. In addition, all plus signs (+) are replaced with spaces.

Standards and compatibility
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also
- “HTML_ENCODE function [HTTP]” on page 182
- “HTTP_ENCODE function [HTTP]” on page 185

HTTP_ENCODE function [HTTP]

Function
Encodes special characters in strings for use with HTTP.

Syntax
```
HTML_ENCODE ( string )
```

Note

Parameters
- **string** Arbitrary string to be used in an HTTP request.
Alphabetical list of functions

Usage
This function returns the string argument after making the following set of substitutions. In addition, all characters with hexadecimal codes less than 1F or greater than 7E are replaced with %nn, where nn is the character code.

<table>
<thead>
<tr>
<th>Character</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>%20</td>
</tr>
<tr>
<td>&quot;</td>
<td>%22</td>
</tr>
<tr>
<td>#</td>
<td>%23</td>
</tr>
<tr>
<td>&amp;</td>
<td>%26</td>
</tr>
<tr>
<td>,</td>
<td>%2C</td>
</tr>
<tr>
<td>;</td>
<td>%3B</td>
</tr>
<tr>
<td>&lt;</td>
<td>%3C</td>
</tr>
<tr>
<td>&gt;</td>
<td>%3E</td>
</tr>
<tr>
<td>[</td>
<td>%5B</td>
</tr>
<tr>
<td>\</td>
<td>%5C</td>
</tr>
<tr>
<td>]</td>
<td>%5D</td>
</tr>
<tr>
<td>`</td>
<td>%60</td>
</tr>
<tr>
<td>{</td>
<td>%7B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>%7D</td>
</tr>
</tbody>
</table>

Standards and compatibility
- SQL92  Vendor extension
- Sybase  Not supported by Adaptive Server Enterprise

See also
“HTML_ENCODE function [HTTP]” on page 182
“HTTP_DECODE function [HTTP]” on page 185

HTTP_HEADER function [HTTP]
Function
Gets the value of an HTTP header.

Syntax

```
HTTP_HEADER ( field-name )
```

**Note**  

Parameters

- **field-name** The name of an HTTP header field.
CHAPTER 4    SQL Functions

Usage
This function returns the value of the named HTTP header field. It is used when processing an HTTP request through a Web service.

Standards and compatibility
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also
- “HTTP_VARIABLE function [HTTP]” on page 187
- “NEXT_HTTP_HEADER function [HTTP]” on page 210
- “NEXT_HTTP_VARIABLE function [HTTP]” on page 211

**HTTP_VARIABLE function [HTTP]**

Function
Gets the value of an HTTP variable.

Syntax
```
HTML_VARIABLE ( var-name [, instance ], header-field )
```

**Note** CIS functional compensation performance considerations apply. See “Conditions that cause processing by SQL Anywhere” in Chapter 3, “Optimizing Queries and Deletions,” in the Performance and Tuning Guide.

Parameters
- **var-name** The name of the an HTTP variable.
- **instance** If more than one variable has the same name, the instance number of the field instance, or NULL to get the first one. Useful for select lists that permit multiple selections.
- **header-field** In a multipart request, a header field name associated with the named field.

Usage
This function returns the value of the named HTTP variable. It is used when processing an HTTP request through a Web service.

Standards and compatibility
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

Reference: Building Blocks, Tables, and Procedures 187
**IFNULL function [Miscellaneous]**

Function

If the first expression is the NULL value, then the value of the second expression is returned. If the first expression is not NULL, the value of the third expression is returned. If the first expression is not NULL and there is no third expression, then the NULL value is returned.

Syntax

```
IFNULL ( expression1, expression2 [ , expression3 ] )
```

Parameters

- **expression1** The expression to be evaluated. Its value determines whether `expression2` or `expression3` is returned.
- **expression2** The return value if `expression1` is NULL.
- **expression3** The return value if `expression1` is not NULL.

Examples

The following statement returns the value -66:

```
SELECT IFNULL( NULL, -66 ) FROM iq_dummy
```

The following statement returns NULL, because the first expression is not NULL and there is no third expression:

```
SELECT IFNULL( -66, -66 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92** Transact-SQL extension
- **Sybase** Not supported by Adaptive Server Enterprise

---

**INDEX_COL function [System]**

Function

Returns the name of the indexed column.

Syntax

```
INDEX_COL ( table-name, index-id, key_# [ , user-id ] )
```

Parameters

- **table-name** A table name.
- **index-id** The index ID of an index of `table-name`.
- **key_#** A key in the index specified by `index-id`. This parameter specifies the column number in the index. For a single column index, `key_#` is equal to 0. For a multicoloumn index, `key_#` is equal to 0 for the first column, 1 for the second column, and so on.
- **user-id** The user ID of the owner of `table-name`. If `user-id` is not specified, this value defaults to the caller’s user ID.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Adaptive Server Enterprise function implemented for Sybase IQ

See also

“OBJECT_ID function [System]” on page 215
**INSERTSTR function [String]**

Function: Inserts a string into another string at a specified position.

Syntax:  

\[ \text{INSERTSTR} \left( \text{numeric-expression}, \text{string-expression1}, \text{string-expression2} \right) \]

Parameters:

- **numeric-expression**: The position after which \text{string-expression2} is to be inserted. Use zero to insert a string at the beginning.
- **string-expression1**: The string into which \text{string-expression2} is to be inserted.
- **string-expression2**: The string to be inserted.

*Note*: The result data type of an INSERTSTR function is a LONG VARCHAR. If you use INSERTSTR in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set INSERTSTR to the correct data type and size.

See also: “REPLACE function [String]”.

**Example**: The following statement returns the value “backoffice”:

\[ \text{SELECT INSERTSTR} \left( 0, \text{'office '}, \text{'back'} \right) \text{ FROM iq_dummy} \]

**Standards and compatibility**

- **SQL92**: Vendor extension
- **Sybase**: Not supported in Adaptive Server Enterprise. The STUFF function is equivalent and is supported in both Adaptive Server Enterprise and Sybase IQ

**See also**: “STUFF function [String]” on page 264

---

**INTTOHEX function [Data type conversion]**

Function: Returns the hexadecimal equivalent of a decimal integer.

Syntax:  

\[ \text{INTTOHEX} \left( \text{integer-expression} \right) \]

Parameters:

- **integer-expression**: The integer to be converted to hexadecimal.

**Examples**: The following statement returns the value 3B9ACA00:

\[ \text{SELECT INTTOHEX} \left( 1000000000 \right) \text{ FROM iq_dummy} \]

The following statement returns the value 00000002540BE400:

\[ \text{SELECT INTTOHEX} \left( 10000000000 \right) \text{ FROM iq_dummy} \]
Usage

If data conversion of input to INTTOHEX conversion fails, Sybase IQ returns an error, unless the CONVERSION_ERROR option is OFF. In that case, the result is NULL.

**ASE_FUNCTION_BEHAVIOR option**  The database option
ASE_FUNCTION_BEHAVIOR specifies that output of IQ functions, including INTTOHEX and HEXTOINT, be consistent with the output of Adaptive Server Enterprise functions. The default value of ASE_FUNCTION_BEHAVIOR is OFF.

When the ASE_FUNCTION_BEHAVIOR option is disabled (the value is OFF):

- The output of INTTOHEX is compatible with SQL Anywhere.
- Depending on the input, the output of INTTOHEX can be 8 digits or 16 digits and is left padded with zeros; the return data type is VARCHAR.
- The output of INTTOHEX does not have a ‘0x’ or ‘0X’ prefix.
- The input to INTTOHEX can be up to a 64-bit integer.

When the ASE_FUNCTION_BEHAVIOR option is enabled (the value is ON):

- The output of INTTOHEX is compatible with ASE.
- The output of INTTOHEX is always 8 digits and is left-padded with zeros; the return data type is VARCHAR.
- The output of INTTOHEX does not have a ‘0x’ or ‘0X’ prefix.
- Sybase IQ INTTOHEX assumes input is a 32-bit signed integer; a larger value can overflow and a conversion error can result. For example, the statement:

  ```sql
  SELECT INTTOHEX( 1000000000 ) FROM iq_dummy
  ```

  returns the value 3B9ACA00. But the statement:

  ```sql
  SELECT INTTOHEX( 10000000000 ) FROM iq_dummy
  ```

  results in a conversion error.

**Standards and compatibility**

- SQL92  Transact-SQL extension
- Sybase  Compatible with Adaptive Server Enterprise

**See also**

CONVERSION_ERROR option [TSQL] in *Reference: Statements and Options*

ASE_FUNCTION_BEHAVIOR option in *Reference: Statements and Options*

“HEXTOINT function [Data type conversion]” on page 179
ISDATE function [Date and time]

Function
Tests whether a string argument can be converted to a date. If a conversion is possible, the function returns 1; otherwise, it returns 0. If the argument is null, 0 is returned.

Syntax
ISDATE ( string )

Parameters
string The string to be analyzed to determine whether the string represents a valid date.

Example
The following example tests whether the birth_date column holds valid dates, returning invalid dates as NULL, and valid dates in date format.

```sql
select birth_date from MyData;
-----------------------------
1990/32/89
0101/32/89
1990/12/09

select
    case when isdate(birth_date)=0 then NULL
    else cast(birth_date as date)
    end
from MyData;
------------------------------------
(NULL)
(NULL)
1990-12-09
```

Standards and compatibility
- SQL92  Vendor extension
- SQL99  Vendor extension
- Sybase  Not supported by Adaptive Server Enterprise

ISNULL function [Miscellaneous]

Function
Returns the value of the first non-NULL expression in the parameter list.

Syntax
ISNULL ( expression, expression [ ..., expression ] )

Parameters
expression An expression to be tested against NULL.

At least two expressions must be passed to the function.

Example
The following statement returns the value -66:

```sql
SELECT ISNULL( NULL, -66, 55, 45, NULL, 16 ) FROM iq_dummy
```
Alphabetical list of functions

Usage
The ISNULL function is the same as the COALESCE function.

Standards and compatibility
• SQL92 Transact-SQL extension
• Sybase Not supported by Adaptive Server Enterprise

See also
“COALESCE function [Miscellaneous]” on page 137

ISNUMERIC function [Miscellaneous]

Function
Tests whether a string argument can be converted to a numeric. If a conversion is possible, the function returns 1; otherwise, it returns 0. If the argument is null, 0 is returned.

Syntax
ISNUMERIC ( string )


Parameters
string The string to be analyzed to determine whether the string represents a valid numeric value.

Usage
For optimal performance, avoid using ISNUMERIC in predicates, where it is processed by the SQL Anywhere portion of the product and cannot take advantage of the performance features of Sybase IQ.

Example
The following example tests whether the height_in_cms column holds valid numeric data, returning invalid numeric data as NULL, and valid numeric data in int format.

```
data height_in_cms
------------------------
asde asde 180 156

select case
    when isnumeric(height_in_cms)=0 then NULL
    else cast(height_in_cms as int)
end
from MyData
```

Standards and compatibility
• SQL92 Vendor extension
LAST_VALUE function [Aggregate]

Function
Returns the last value from a set of values.

Syntax
LAST_VALUE (expression [IGNORE NULLS | RESPECT NULLS])
OVER (window-spec)

Parameters
expression The expression on which to determine the last value in an ordered set.

Usage
LAST_VALUE returns the last value in a set of values, which is usually an ordered set. If the last value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. If you specify IGNORE NULLS, then LAST_VALUE returns the last non-null value in the set, or NULL if all values are null.

The data type of the returned value is the same as that of the input value.

You cannot use LAST_VALUE or any other analytic function for expression. That is, you cannot nest analytic functions, but you can use other built-in function expressions for expression.

If the window-spec does not contain an ORDER BY then the result is arbitrary. If there is no window-spec, then the result is arbitrary.

You can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Note DISTINCT is not supported.

Example
The following example returns the salary of each employee, plus the name of the employee with the highest salary in their department:

```sql
SELECT GivenName + ' ' + Surname AS employee_name,
       Salary, DepartmentID,
       LAST_VALUE( employee_name ) OVER Salary_Window AS highest_paid
FROM Employees
WINDOW Salary_Window AS ( PARTITION BY DepartmentID
                          ORDER BY Salary
                          RANGE BETWEEN UNBOUNDED PRECEDING
```
The returned result set is:

<table>
<thead>
<tr>
<th>employee_name</th>
<th>Salary</th>
<th>DepartmentID</th>
<th>highest_paid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael Lynch</td>
<td>24903.000</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Joseph Barker</td>
<td>27290.000</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Sheila Romero</td>
<td>27500.000</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Felicia Kuo</td>
<td>28200.000</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Jeannette Bertrand</td>
<td>29800.000</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Jane Braun</td>
<td>34300.000</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Anthony Rebeiro</td>
<td>34576.000</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Charles Crowley</td>
<td>41700.000</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Jose Martinez</td>
<td>55500.800</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Doug Charlton</td>
<td>28300.000</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Elizabeth Lambert</td>
<td>29384.000</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Joyce Butterfield</td>
<td>34011.000</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Robert Nielsen</td>
<td>34889.000</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Alex Ahmed</td>
<td>34992.000</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Ruth Wetherby</td>
<td>35745.000</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Standards and compatibility

- **SQL2008**  SQL/OLAP feature T612
- **Sybase**  Compatible with SQL Anywhere

**LCASE function [String]**

Function: Converts all characters in a string to lowercase.

Syntax: `LCASE ( string-expression )`
Parameters

**string-expression**  The string to be converted to lowercase.

**Note** The result data type of an LCASE function is a LONG VARCHAR. If you use LCASE in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set LCASE to the correct data type and size.

See “REPLACE function [String]” for more information.

Example

The following statement returns the value “lower case”:

```
SELECT LCASE ( 'LOWER CasE' ) FROM iq_dummy
```

Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  LCASE is not supported in Adaptive Server Enterprise; you can use LOWER to get the same functionality

See also

- “LOWER function [String]” on page 200
- “UCASE function [String]” on page 329
- “UPPER function [String]” on page 329

**LEFT function [String]**

**Function**  Returns a specified number of characters from the beginning of a string.

**Syntax**  

```
LEFT ( string-expression, numeric-expression )
```

**Parameters**

- **string-expression**  The string.
- **numeric-expression**  The number of characters to return.

**Example**

The following statement returns the value “choco”:

```
SELECT LEFT ( 'chocolate', 5 ) FROM iq_dummy
```
Alphabetical list of functions

LEN function [String]

Function

Takes one argument as an input of type BINARY or STRING and returns the number of characters, as defined by the database's collation sequence, of a specified string expression, excluding trailing blanks. The result may differ from the string's byte length for multi-byte character sets.

BINARY and VARBINARY are also allowed, in which case LEN() returns the number of bytes of the input.

LEN is an alias of LENGTH function.

Syntax

LEN ( string_expr )

Parameters

string_expr is the string expression to be evaluated.

Example

The following example returns the value 3152:

```
select len(Photo) from Products
where ID = 500
```

Usage

This function is the equivalent of CHAR_LENGTH ( string_expression ).

Permissions

Any user can execute LEN.

Standards and compatibility

ANSI SQL – Compliance level: Transact-SQL extension
See also  Data types  
CHAR, NCHAR, VARCHAR, NVARCHAR.

See Chapter 3, “SQL Data Types.”

Functions  “CHAR_LENGTH function [String]” on page 136 and “STR_REPLACE function [String]” on page 261.

For general information about string functions, see “String functions” on page 115.

**LENGTH function [String]**

Function  Returns the number of characters in the specified string.

Syntax  
```
LENGTH ( string-expression )
```

Parameters  
string-expression  The string.

Example  
The following statement returns the value 9:
```
SELECT LENGTH( 'chocolate' ) FROM iq_dummy
```

Usage  
If the string contains multibyte characters, and the proper collation is being used, LENGTH returns the number of characters, not the number of bytes. If the string is of BINARY data type, the LENGTH function behaves as BYTE_LENGTH.

The LENGTH function is the same as the CHAR_LENGTH function.

Standards and compatibility  
- **SQL92**  Vendor extension
- **Sybase**  Not supported by Adaptive Server Enterprise. Use the CHAR_LENGTH function instead

See also  
“BYTE_LENGTH function [String]” on page 133

“CHAR_LENGTH function [String]” on page 136

Chapter 11, “International Languages and Character Sets” in the System Administration Guide: Volume I

**LN function [Numeric]**

Function  Returns the natural logarithm of the specified expression.

Syntax  
```
LN ( numeric-expression )
```
### Alphabetical list of functions

#### Parameters

| **expression** | Is a column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, the LN function generates an error. The return value is of DOUBLE data type. |

#### Usage

LN takes one argument. For example, LN (20) returns 2.995732.

The LN function is an alias of the LOG function.

#### Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise. Use the LOG function instead

#### See also

“LOG function [Numeric]” on page 199

Chapter 11, “International Languages and Character Sets” in the System Administration Guide: Volume 1

---

### LOCATE function [String]

**Function**

Returns the position of one string within another.

**Syntax**

```sql
LOCATE( string-expression1, string-expression2 [ , numeric-expression ] )
```

**Parameters**

- **string-expression1** The string to be searched.
- **string-expression2** The string for which you are searching. This string is limited to 255 bytes.
- **numeric-expression** The character position at which to begin the search in the string. The first character is position 1. If the starting offset is negative, LOCATE returns the last matching string offset, rather than the first. A negative offset indicates how much of the end of the string to exclude from the search. The number of bytes excluded is calculated as ( -1 * offset ) - 1.

The **numeric-expression** is a 32 bit signed integer for CHAR, VARCHAR, and BINARY columns.

**Examples**

The following statement returns the value 8:

```sql
SELECT LOCATE( 'office party this week – rsvp as soon as possible', 'party', 2 ) FROM iq_dummy
```

In the second example, the **numeric-expression** starting offset for the search is a negative number.

```sql
CREATE TABLE t1(name VARCHAR(20), dirname VARCHAR(60));
INSERT INTO t1
```
VALUES('m1000','c:\test\functions\locate.sql');
INSERT INTO t1
VALUES('m1001','d:\test\functions\trim.sql');
COMMIT;

SELECT LOCATE(dirname, '\', -1), dirname FROM t1;
The result is:
18 c:\test\functions\locate.sql
18 d:\test\functions\trim.sql

Usage
If numeric-expression is specified, the search starts at that offset into the string
being searched.

If numeric-expression is not specified, LOCATE returns only the position of the
first instance of the specified string.

The first string can be a long string (longer than 255 bytes), but the second is
limited to 255 bytes. If a long string is given as the second argument, the
function returns a NULL value.

If any of the arguments is NULL, the result is NULL.

Searching for a zero-length string returns 1.

If the string does not contain the specified string, the LOCATE function returns
zero (0).

All the positions or offsets, returned or specified, in the LOCATE function are
always character offsets and may be different from the byte offset for multibyte
data.

Standards and
compatibility

• SQL92 Vendor extension
• Sybase Not supported by Adaptive Server Enterprise

See also
Chapter 4, “Function Support” in Large Objects Management in Sybase IQ

LOG function [Numeric]

Function
Returns the natural logarithm of a number.

LN is an alias of LOG.

Syntax
LOG ( numeric-expression )

Parameters
numeric-expression The number.

Example
The following statement returns the value 3.912023:
Alphabetical list of functions

SELECT LOG( 50 ) FROM iq_dummy

Standards and compatibility

• SQL92  Vendor extension
• Sybase  Compatible with Adaptive Server Enterprise

See also

“LOG10 function [Numeric]” on page 200

LOG10 function [Numeric]

Function

Returns the base 10 logarithm of a number.

Syntax

LOG10( numeric-expression )

Parameters

numeric-expression  The number.

Example

The following statement returns the value 1.698970.

SELECT LOG10( 50 ) FROM iq_dummy

Standards and compatibility

• SQL92  Vendor extension
• Sybase  Compatible with Adaptive Server Enterprise

See also

“LOG function [Numeric]” on page 199

LOWER function [String]

Function

Converts all characters in a string to lowercase.

Syntax

LOWER( string-expression )

Parameters

string-expression  The string to be converted.

Note  The result data type of a LOWER function is a LONG VARCHAR. If you use LOWER in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set LOWER to the correct data type and size.

See “REPLACE function [String]” for more information.

Example

The following statement returns the value “lower case”:

SELECT LOWER( 'LOWER CasE' ) FROM iq_dummy

Standards and compatibility

• SQL92  SQL92 compatible.
• Sybase  Compatible with Adaptive Server Enterprise
LTRIM function [String]

Function
Removes leading blanks from a string.

Syntax
LTRIM ( string-expression )

Parameters
string-expression The string to be trimmed.

Note The result data type of an LTRIM function is a LONG VARCHAR. If you use LTRIM in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set LTRIM to the correct data type and size.

See “REPLACE function [String]” on page 242 for more information.

Example
The following statement returns the value “Test Message” with all leading blanks removed:

SELECT LTRIM ( ' Test Message' ) FROM iq_dummy

Standards and compatibility
• SQL92 Vendor extension
• Sybase Compatible with Adaptive Server Enterprise

See also
“RTRIM function [String]” on page 247
“TRIM function [String]” on page 268

MAX function [Aggregate]

Function
Returns the maximum expression value found in each group of rows.

Syntax
MAX ( expression | DISTINCT column-name )

Parameters
expression The expression for which the maximum value is to be calculated. This is commonly a column name.
DISTINCT column-name Returns the same as MAX ( expression ), and is included for completeness.
Example
The following statement returns the value 138948.000, representing the maximum salary in the Employees table:

```
SELECT MAX ( Salary )
FROM Employees
```

Usage
Rows where expression is NULL are ignored. Returns NULL for a group containing no rows.

Standards and compatibility
• SQL92 SQL92 compatible.
• Sybase Compatible with Adaptive Server Enterprise.

See also
“MIN function [Aggregate]” on page 203

MEDIAN function [Aggregate]
Function
Returns the median of an expression.

Syntax 1
```
MEDIAN([ALL | DISTINCT] expression)
```

Syntax 2
```
MEDIAN([ALL | DISTINCT] expression)
OVER (window-spec)
```

`window-spec`: See Syntax 2 instructions in the Usage section, below.

Parameters
expression A numeric expression for which a median value is to be computed.

Usage
The median is the number separating the higher half of a sample, a population, or a probability distribution, from the lower half.

The data type of the returned value is the same as that of the input value. NULLs are ignored in the calculation of the median value. You can use the optional keyword DISTINCT to eliminate duplicate values before the aggregate function is applied. ALL, which performs the operation on all rows, is the default.

**Note** ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1.
Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

**Note** The window-spec cannot contain a ROW, RANGE or ORDER BY specification; window-spec can only specify a PARTITION clause. DISTINCT is not supported if a WINDOW clause is used.

**Example**

The following query returns the median salary for each department in Florida:

```sql
SELECT DepartmentID, Surname, Salary,
       MEDIAN(Salary) OVER (PARTITION BY DepartmentID)
          "Median"
FROM Employees
WHERE State IN ('FL')
```

The returned result is:

Table 4-24: MEDIAN result set

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>Surname</th>
<th>Salary</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Lull</td>
<td>87900.000</td>
<td>73870.000</td>
</tr>
<tr>
<td>100</td>
<td>Gowda</td>
<td>59840.000</td>
<td>73870.000</td>
</tr>
<tr>
<td>200</td>
<td>Sterling</td>
<td>64900.000</td>
<td>76200.000</td>
</tr>
<tr>
<td>200</td>
<td>Kelly</td>
<td>87500.000</td>
<td>76200.000</td>
</tr>
<tr>
<td>300</td>
<td>Litton</td>
<td>58930.000</td>
<td>58930.000</td>
</tr>
<tr>
<td>400</td>
<td>Francis</td>
<td>53870.000</td>
<td>3870.000</td>
</tr>
<tr>
<td>400</td>
<td>Charlton</td>
<td>28300.000</td>
<td>53870.000</td>
</tr>
<tr>
<td>400</td>
<td>Evans</td>
<td>68940.000</td>
<td>53870.000</td>
</tr>
</tbody>
</table>

**Standards and compatibility**

- SQL2008  Vendor extension

**MIN function [Aggregate]**

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns the minimum expression value found in each group of rows.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>MIN ( expression</td>
</tr>
<tr>
<td>Parameters</td>
<td>expression The expression for which the minimum value is to be calculated. This is commonly a column name.</td>
</tr>
</tbody>
</table>
Alphabetical list of functions

DISTINCT column-name  Returns the same as MIN ( expression ), and is included for completeness.

Example  The following statement returns the value 24903.000, representing the minimum salary in the Employees table:

```
SELECT MIN ( Salary )
FROM Employees
```

Usage  Rows where expression is NULL are ignored. Returns NULL for a group containing no rows.

Standards and compatibility

- SQL92  SQL92 compatible.
- Sybase  Compatible with Adaptive Server Enterprise.

See also  “MAX function [Aggregate]” on page 201


MINUTE function [Date and time]

Function  Returns a number from 0 to 59 corresponding to the minute component of the specified date/time value.

Syntax  `MINUTE ( datetime-expression )`

Parameters  `datetime-expression`  The date/time value.

Example  The following statement returns the value 22:

```
SELECT MINUTE( '1998-07-13 12:22:34' ) FROM iq_dummy
```

Standards and compatibility

- SQL92  Vendor extension
- Sybase  Compatible with Adaptive Server Enterprise

MINUTES function [Date and time]

Function  Returns the number of minutes since an arbitrary date and time, the number of whole minutes between two specified times, or adds the specified integer-expression number of minutes to a time.

Syntax  `MINUTES ( datatime-expression
| datatime-expression, datatime-expression
| datatime-expression, integer-expression )`

Parameters  `datetime-expression`  A date and time.
integer-expression The number of minutes to be added to the datetime-expression. If integer-expression is negative, the appropriate number of minutes are subtracted from the date/time. If you supply an integer expression, the datetime-expression must be explicitly cast as a datetime data type.

For information on casting data types, see “CAST function [Data type conversion]” on page 133.

Examples

The following statement returns the value 1051125487:

```sql
SELECT MINUTES( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 240, to signify the difference between the two times:

```sql
SELECT MINUTES( '1999-07-13 06:07:12',
                 '1999-07-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the datetime value 1999-05-12 21:10:07.000:

```sql
SELECT MINUTES( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5) FROM iq_dummy
```

Usage

The second syntax returns the number of whole minutes from the first date/time to the second date/time. The number might be negative.

Standards and compatibility

• **SQL92** Vendor extension
• **Sybase** Not supported by Adaptive Server Enterprise

MOD function [Numeric]

Function

Returns the remainder when one whole number is divided by another.

Syntax

```sql
MOD( dividend, divisor )
```

Parameters

- **dividend** The dividend, or numerator of the division.
- **divisor** The divisor, or denominator of the division.

Example

The following statement returns the value 2:

```sql
SELECT MOD( 5, 3 ) FROM iq_dummy
```

Usage

Division involving a negative dividend gives a negative or zero result. The sign of the divisor has no effect.

Standards and compatibility

• **SQL92** Vendor extension
• **Sybase** Not supported in Adaptive Server Enterprise. The % operator is used as a modulo operator in Adaptive Server Enterprise
Alphabetical list of functions

See also

“REMAINDER function [Numeric]” on page 241

MONTH function [Date and time]
Function Returns a number from 1 to 12 corresponding to the month of the given date.
Syntax
```
MONTH ( date-expression )
```
Parameters
date-expression A date/time value.
Example
The following statement returns the value 7:
```
SELECT MONTH( '1998-07-13' ) FROM iq_dummy
```
Standards and compatibility
• SQL92 Vendor extension
• Sybase Not supported by Adaptive Server Enterprise

MONTHNAME function [Date and time]
Function Returns the name of the month from the specified date expression.
Syntax
```
MONTHNAME ( date-expression )
```
Parameters
date-expression The datetime value.
Example
The following statement returns the value September, when the DATE_ORDER option is set to the default value of ymd.
```
SELECT MONTHNAME( '1998-09-05' ) FROM iq_dummy
```
Standards and compatibility
• SQL92 Vendor extension
• Sybase Not supported by Adaptive Server Enterprise

MONTHS function [Date and time]
Function Returns the number of months since an arbitrary starting date/time or the number of months between two specified date/times, or adds the specified integer-expression number of months to a date/time.
Syntax
```
MONTHS ( date-expression
| date-expression, datetime-expression
| date-expression, integer-expression )
```
Parameters

date-expression  A date and time.

integer-expression  The number of months to be added to the date-expression. If integer-expression is negative, the appropriate number of months are subtracted from the date/time value. If you supply an integer expression, the date-expression must be explicitly cast as a datetime data type.

For information on casting data types, see the section “CAST function [Data type conversion]” on page 133.

Examples

The following statement returns the value 23982:

   SELECT MONTHS( '1998-07-13 06:07:12' ) FROM iq_dummy

The following statement returns the value 2, to signify the difference between the two dates:

   SELECT MONTHS( '1999-07-13 06:07:12',
                 '1999-09-13 10:07:12' ) FROM iq_dummy

The following statement returns the datetime value 1999-10-12 21:05:07.000:

   SELECT MONTHS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5) FROM iq_dummy

Usage

The first syntax returns the number of months since an arbitrary starting date. This number is often useful for determining whether two date/time expressions are in the same month in the same year.

   MONTHS( invoice_sent ) = MONTHS( payment_received )

Note that comparing the MONTH function would incorrectly include a payment made 12 months after the invoice was sent.

The second syntax returns the number of months from the first date to the second date. The number might be negative. It is calculated from the number of the first days of the month between the two dates. Hours, minutes and seconds are ignored.

The third syntax adds integer-expression months to the given date. If the new date is past the end of the month (such as MONTHS( '1992-01-31', 1 )) the result is set to the last day of the month. If integer-expression is negative, the appropriate number of months are subtracted from the date. Hours, minutes and seconds are ignored.

Standards and compatibility

- SQL92  Vendor extension
- Sybase  Not supported by Adaptive Server Enterprise
NEWID function [Miscellaneous]

Function
Generates a UUID (Universally Unique Identifier) value. The returned UUID value is a binary. A UUID is the same as a GUID (Globally Unique Identifier).

Syntax
NEWID()

Parameters
There are no parameters associated with NEWID().

Example
The following statement creates the table t1 and then updates the table, setting the value of the column uid_col to a unique identifier generated by the NEWID function, if the current value of the column is NULL.

```
CREATE TABLE t1 (uid_col int);
UPDATE t1
  SET uid_col = NEWID()
  WHERE uid_col IS NULL
```

If you execute the following statement,

```
SELECT NEWID()
```

the unique identifier is returned as a BINARY(16). For example, the value might be 0xd3749fe09cf446e399913bc6434f1f08. You can convert this string into a readable format using the UUIDTOSTR() function.

Usage
The NEWID() function generates a unique identifier value. UUIDs can be used to uniquely identify objects in a database. The values are generated such that a value produced on one computer does not match that produced on another, hence they can also be used as keys in replication and synchronization environments.

The NEWID function is supported only in the following positions:
- SELECT list of a top level query block
- SET clause of an UPDATE statement
- VALUES clause of INSERT...VALUES

You can use a value generated by the NEWID function as a column default value in a Sybase IQ table.

Standards and compatibility
- SQL2 Vendor extension
- SQL99 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise.

See also
“STRTOUUID function [String]” on page 263
“UUIDTOSTR function [String]” on page 331
“Binary data types” on page 77
“Character data types” on page 69

**NEXT_CONNECTION** function [System]

**Function**
Returns the next connection number, or the first connection if the parameter is NULL.

**Syntax**

```
NEXT_CONNECTION ( { NULL | connection-id } )
```

**Note**

**Parameters**

- **connection-id** An integer, usually returned from a previous call to NEXT_CONNECTION. If connection-id is NULL, NEXT_CONNECTION returns the first connection ID.

**Examples**
The following statement returns an identifier for the first connection. The identifier is an integer value like 569851433.

```
SELECT NEXT_CONNECTION( NULL ) FROM iq_dummy
```

The following statement returns a value like 1661140050:

```
SELECT NEXT_CONNECTION( 569851433 ) FROM iq_dummy
```

**Usage**
You can use NEXT_CONNECTION to enumerate the connections to a database. To get the first connection, pass NULL; to get each subsequent connection, pass the previous return value. The function returns NULL when there are no more connections.

**Standards and compatibility**

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

**NEXT_DATABASE** function [System]

**Function**
Returns the next database ID number, or the first database if the parameter is NULL.
Alphabetical list of functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT_DATABASE</td>
<td>Gets the next database name.</td>
</tr>
<tr>
<td>NEXT_HTTP_HEADER</td>
<td>Gets the next HTTP header name.</td>
</tr>
</tbody>
</table>

### NEXT_DATABASE function

**Syntax**

\[
\text{NEXT_DATABASE}\left\{\{\text{NULL} \mid \text{database-id}\}\right\}
\]

**Note**


**Parameters**

- **database-id** An integer that specifies the ID number of the database.

**Examples**

The following statement returns the value 0, the first database value:

```
SELECT NEXT_DATABASE( NULL ) FROM iq_dummy
```

The following statement returns NULL, indicating that there are no more databases on the server:

```
SELECT NEXT_DATABASE( 0 ) FROM iq_dummy
```

**Usage**

You can use NEXT_DATABASE to enumerate the databases running on a database server. To get the first database, pass NULL; to get each subsequent database, pass the previous return value. The function returns NULL when there are no more databases.

**Standards and compatibility**

- SQL2 Transact-SQL extension.
- Sybase Not supported by Adaptive Server Enterprise.

**See also**

“OBJECT_ID function [System]” on page 215

### NEXT_HTTP_HEADER function [HTTP]

**Function**

Gets the next HTTP header name.

**Syntax**

\[
\text{NEXT_HTTP_HEADER}\left(\text{header-name}\right)
\]

**Note**


**Parameters**

- **header-name** The name of the previous header. If header-name is null, this function returns the name of the first HTTP header.

**Usage**

This function iterates over the HTTP headers included within a request. Calling it with NULL causes it to return the name of the first header. Subsequent headers are retrieved by passing the function the name of the previous header. This function returns NULL when called with the name of the last header.
Calling this function repeatedly returns all the header fields exactly once, but not necessarily in the order in which they appear in the HTTP request.

**Standards and compatibility**
- SQL92  Vendor extension
- Sybase  Not supported by Adaptive Server Enterprise

### NEXT_HTTP_VARIABLE function [HTTP]

**Function**
Get the next HTTP variable name.

**Syntax**
```
NEXT_HTTP_VARIABLE ( var-name )
```

**Note**

**Parameters**
*var-name*  The name of the previous variable. If var-name is null, this function returns the name of the first HTTP variable.

**Usage**
This function iterates over the HTTP variables included within a request. Calling it with NULL causes it to return the name of the first variable. Subsequent variables are retrieved by passing the function the name of the previous variable. This function returns NULL when called with the name of the final variable.

Calling this function repeatedly returns all the variables exactly once, but not necessarily in the order in which they appear in the HTTP request. The variables `url` or `url1`, `url2`, …, `url10` are included if URL PATH is set to ON or ELEMENTS, respectively.

**Standards and compatibility**
- SQL92  Vendor extension
- Sybase  Not supported by Adaptive Server Enterprise

**See also**
- “HTML_DECODE function [HTTP]” on page 181
- “HTTP_VARIABLE function [HTTP]” on page 187
- “NEXT_HTTP_HEADER function [HTTP]” on page 210

### NOW function [Date and time]

**Function**
Returns the current date and time. This is the historical syntax for `CURRENT_TIMESTAMP`.

Reference: Building Blocks, Tables, and Procedures 211
### NOW

**Syntax**

`NOW(*)`

**Example**

The following statement returns the current date and time.

```sql
SELECT NOW(*) FROM iq_dummy
```

### NTILE function [Analytical]

**Function**

Distributes query results into a specified number of “buckets” and assigns the bucket number to each row in the bucket.

**Syntax**

```sql
NTILE ( expression1 )
OVER ( ORDER BY expression2 [ ASC | DESC ] )
```

**Parameters**

- `expression1`: A constant integer from 1 to 32767, which specifies the number of buckets.
- `expression2`: A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

**Example**

The following example uses the `NTILE` function to determine the sale status of car dealers. The dealers are divided into four groups based on the number of cars each dealer sold. The dealers with `ntile = 1` are in the top 25% for car sales.

```sql
SELECT dealer_name, sales,
NTILE(4) OVER ( ORDER BY sales DESC )
FROM carSales;
```

<table>
<thead>
<tr>
<th>dealer_name</th>
<th>sales</th>
<th>ntile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boston</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>Worcester</td>
<td>950</td>
<td>1</td>
</tr>
<tr>
<td>Providence</td>
<td>950</td>
<td>1</td>
</tr>
<tr>
<td>SF</td>
<td>940</td>
<td>1</td>
</tr>
<tr>
<td>Lowell</td>
<td>900</td>
<td>2</td>
</tr>
<tr>
<td>Seattle</td>
<td>900</td>
<td>2</td>
</tr>
<tr>
<td>Natick</td>
<td>870</td>
<td>2</td>
</tr>
<tr>
<td>New Haven</td>
<td>850</td>
<td>2</td>
</tr>
<tr>
<td>Portland</td>
<td>800</td>
<td>3</td>
</tr>
<tr>
<td>Houston</td>
<td>780</td>
<td>3</td>
</tr>
<tr>
<td>Hartford</td>
<td>780</td>
<td>3</td>
</tr>
<tr>
<td>Dublin</td>
<td>750</td>
<td>3</td>
</tr>
<tr>
<td>Austin</td>
<td>650</td>
<td>4</td>
</tr>
<tr>
<td>Dallas</td>
<td>640</td>
<td>4</td>
</tr>
</tbody>
</table>
To find the top 10% of car dealers by sales, you specify `NTILE(10)` in the example `SELECT` statement. Similarly, to find the top 50% of car dealers by sales, specify `NTILE(2)`.

**Usage**

`NTILE` is a rank analytical function that distributes query results into a specified number of buckets and assigns the bucket number to each row in the bucket. You can divide a result set into one-hundredths (percentiles), tenths (deciles), fourths (quartiles), or other numbers of groupings.

`NTILE` requires an `OVER (ORDER BY)` clause. The `ORDER BY` clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. Note that this `ORDER BY` clause is used only within the `OVER` clause and is *not* an `ORDER BY` for the `SELECT`. No aggregation functions in the rank query are allowed to specify `DISTINCT`.

The `OVER` clause indicates that the function operates on a query result set. The result set is the rows that are returned after the `FROM`, `WHERE`, `GROUP BY`, and `HAVING` clauses have all been evaluated. The `OVER` clause defines the data set of the rows to include in the computation of the rank analytical function.

The `ASC` or `DESC` parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

`NTILE` is allowed only in the select list of a `SELECT` or `INSERT` statement or in the `ORDER BY` clause of the `SELECT` statement. `NTILE` can be in a view or a union. The `NTILE` function cannot be used in a subquery, a `HAVING` clause, or in the select list of an `UPDATE` or `DELETE` statement. Only one `NTILE` function is allowed per query.

**Standards and compatibility**

- SQL92 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise or SQL Anywhere.

**See also**

“Analytical functions” on page 104

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

---

### NULLIF function [Miscellaneous]

**Function**

Provides an abbreviated `CASE` expression by comparing expressions.

**Syntax**

```
NULLIF ( expression1, expression2 )
```

**Parameters**

- `expression1` An expression to be compared.
### expression2

An expression to be compared.

**Examples**
The following statement returns a:

```sql
SELECT NULLIF( 'a', 'b' ) FROM iq_dummy
```

The following statement returns NULL:

```sql
SELECT NULLIF( 'a', 'a' ) FROM iq_dummy
```

**Usage**

NULLIF compares the values of the two expressions.

- If the first expression equals the second expression, NULLIF returns NULL.
- If the first expression does not equal the second expression, or if the second expression is NULL, NULLIF returns the first expression.

The NULLIF function provides a short way to write some CASE expressions. NULLIF is equivalent to:

```sql
CASE WHEN expression1 = expression2 THEN NULL
ELSE expression1 END
```

**Standards and compatibility**
- **SQL92** Transact-SQL extension.
- **Sybase** Not supported by Adaptive Server Enterprise.

**See also**

“CASE expressions” on page 29

### NUMBER function [Miscellaneous]

**Function**

Generates numbers starting at 1 for each successive row in the results of the query.

**Syntax**

```sql
NUMBER ( * )
```

**Example**
The following statement returns this numbered list:

```sql
number(*)
```

```
1
2
3
4
5
```

```sql
SELECT NUMBER( * )
FROM Departments
WHERE DepartmentID > 10
```
Usage

Use the \texttt{NUMBER} function only in a select list or a \texttt{SET} clause of an \texttt{UPDATE} statement. For example, the following statement updates each row of the \texttt{seq\_id} column with a number 1 greater than the previous row. The number is applied in the order specified by the \texttt{ORDER BY} clause.

\begin{verbatim}
update empl
  set seq\_id = number(*)
order by empl\_id
\end{verbatim}

In an \texttt{UPDATE} statement, if the \texttt{NUMBER(*)} function is used in the \texttt{SET} clause and the \texttt{FROM} clause specifies a one-to-many join, \texttt{NUMBER(*)} generates unique numbers that increase, but may not increment sequentially due to row elimination.

\texttt{NUMBER} can also be used to generate primary keys when using the \texttt{INSERT} from \texttt{SELECT} statement, although using \texttt{IDENTITY/AUTOINCREMENT} is a preferred mechanism for generating sequential primary keys.

\textbf{Note} A syntax error is generated if you use \texttt{NUMBER} in a \texttt{DELETE} statement, \texttt{WHERE} clause, \texttt{HAVING} clause, \texttt{ORDER BY} clause, subquery, query involving aggregation, any constraint, \texttt{GROUP BY}, \texttt{DISTINCT}, a query containing \texttt{UNION ALL}, or a derived table.

\begin{tabular}{|l|}
\hline
\textbf{Standards and compatibility} & \textbullet SQL92 Vendor extension \\
\hline
\textbullet Sybase Not supported by Adaptive Server Enterprise. \\
\hline
\end{tabular}

\textbf{OBJECT\_ID function [System]}

\textbf{Function} Returns the object ID.

\textbf{Syntax} \begin{verbatim}
OBJECT\_ID ( object\_name )
\end{verbatim}

\textbf{Parameters} \texttt{object\_name} The name of the object.

\textbf{Examples} The following statement returns the object ID 100209 of the \textit{Customers} table:

\begin{verbatim}
SELECT OBJECT\_ID ('CUSTOMERS') FROM iq\_dummy
\end{verbatim}

\begin{tabular}{|l|}
\hline
\textbf{Standards and compatibility} & \textbullet SQL92 Vendor extension \\
\textbullet Sybase Adaptive Server Enterprise function implemented for Sybase IQ. \\
\hline
\end{tabular}

\textbf{See also} “COL\_NAME function [System]” on page 138

“DB\_ID function [System]” on page 161

Reference: Building Blocks, Tables, and Procedures
OBJECT_NAME function [System]

Function
Returns the object name.

Syntax
OBJECT_NAME ( object-id [ , database-id ] )

Parameters
object-id The object ID.
database-id The database ID.

Examples
The following statement returns the name “customer:”

SELECT OBJECT_NAME ( 100209 ) FROM iq_dummy

Standards and compatibility
• SQL92 Vendor extension
• Sybase Adaptive Server Enterprise function implemented for Sybase IQ.

See also
“COL_NAME function [System]” on page 138
“DB_NAME function [System]” on page 162
“OBJECT_ID function [System]” on page 215

OCTET_LENGTH function [String]

Function
Returns an unsigned 64-bit value containing the byte length of the column parameter.

Syntax
OCTET_LENGTH( column-name )

Parameters
column-name The name of a column.

Usage
The return value of a NULL argument is NULL.

The OCTET_LENGTH function supports all Sybase IQ data types.

Standards and compatibility
Sybase Not supported by SQL Anywhere or Adaptive Server Enterprise.

See also
“BIT_LENGTH function [String]” on page 132
**PATINDEX function [String]**

_Returns the starting position of the first occurrence of a specified pattern._

**Syntax**

```
PATINDEX ( '%pattern%', string-expression )
```

**Parameters**

- **pattern**  
The pattern for which you are searching. This string is limited to 126 bytes for patterns with wildcards. If the leading percent wildcard is omitted, PATINDEX returns one (1) if the pattern occurs at the beginning of the string, and zero if not. If _pattern_ starts with a percent wildcard, then the two leading percent wildcards are treated as one.

The pattern uses the same wildcards as the LIKE comparison. Table 4-25 lists the pattern wildcards.

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ (underscore)</td>
<td>Any one character</td>
</tr>
<tr>
<td>% (percent)</td>
<td>Any string of zero or more characters</td>
</tr>
<tr>
<td>[]</td>
<td>Any single character in the specified range or set</td>
</tr>
<tr>
<td>['']</td>
<td>Any single character not in the specified range or set</td>
</tr>
</tbody>
</table>

Patterns without wildcards (percent % or underscore _) can be up to 255 bytes in length.

**string-expression**  
The string to be searched for the pattern.

**Examples**

The following statement returns the value 2:

```
SELECT PATINDEX( '%hoco%', 'chocolate' ) FROM iq_dummy
```

The following statement returns the value 11:

```
SELECT PATINDEX ( '%4_5_', '0a1A 2a3A 4a5A') FROM iq_dummy
```

**Usage**

PATINDEX returns the starting position of the first occurrence of the pattern. If the string being searched contains more than one instance of the string pattern, PATINDEX returns only the position of the first instance.

If the pattern is not found, PATINDEX returns zero (0).

Searching for a pattern longer than 126 bytes returns NULL.

Searching for a zero-length string returns 1.

If any of the arguments is NULL, the result is zero (0).
All the positions or offsets, returned or specified, in the PATINDEX function are always character offsets and may be different from the byte offset for multibyte data.

PATINDEX returns a 32 bit unsigned integer position for CHAR and VARCHAR columns.

Standards and compatibility

- SQL92  Vendor extension
- Sybase  Compatible with Adaptive Server Enterprise.

See also

“LIKE conditions” on page 40
“LOCATE function [String]” on page 198
Chapter 4, “Function Support” in Large Objects Management in Sybase IQ

PERCENT_RANK function [Analytical]

Function
Computes the (fractional) position of one row returned from a query with respect to the other rows returned by the query, as defined by the ORDER BY clause. Returns a decimal value between 0 and 1.

Syntax
PERCENT_RANK() OVER ( ORDER BY expression [ ASC | DESC ])

Parameters
expression  A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

Example
The following statement illustrates the use of the PERCENT_RANK function:

```sql
SELECT s_suppkey, SUM(s_acctBal) AS sum_acctBal,
PERCENT_RANK() OVER ( ORDER BY SUM(s_acctBal) DESC )
AS percent_rank_all FROM supplier GROUP BY s_suppkey;
```

<table>
<thead>
<tr>
<th>s_suppkey</th>
<th>sum_acctBal</th>
<th>percent_rank_all</th>
</tr>
</thead>
<tbody>
<tr>
<td>supplier#011</td>
<td>200000</td>
<td>0</td>
</tr>
<tr>
<td>supplier#002</td>
<td>200000</td>
<td>0</td>
</tr>
<tr>
<td>supplier#013</td>
<td>123000</td>
<td>0.3333</td>
</tr>
<tr>
<td>supplier#004</td>
<td>110000</td>
<td>0.5</td>
</tr>
<tr>
<td>supplier#035</td>
<td>110000</td>
<td>0.5</td>
</tr>
<tr>
<td>supplier#006</td>
<td>50000</td>
<td>0.8333</td>
</tr>
<tr>
<td>supplier#021</td>
<td>10000</td>
<td>1</td>
</tr>
</tbody>
</table>
Usage

PERCENT_RANK is a rank analytical function. The percent rank of a row \( R \) is defined as the rank of a row in the groups specified in the `OVER` clause minus one divided by the number of total rows in the groups specified in the `OVER` clause minus one. `PERCENT_RANK` returns a value between 0 and 1. The first row has a percent rank of zero.

The `PERCENT_RANK` of a row is calculated as

\[
\frac{Rx - 1}{N_{totalRow} - 1}
\]

where \( Rx \) is the rank position of a row in the group and \( N_{totalRow} \) is the total number of rows in the group specified by the `OVER` clause.

`PERCENT_RANK` requires an `OVER (ORDER BY)` clause. The `ORDER BY` clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This `ORDER BY` clause is used only within the `OVER` clause and is not an `ORDER BY` for the `SELECT`. No aggregation functions in the rank query are allowed to specify `DISTINCT`.

The `OVER` clause indicates that the function operates on a query result set. The result set is the rows that are returned after the `FROM`, `WHERE`, `GROUP BY`, and `HAVING` clauses have all been evaluated. The `OVER` clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

`PERCENT_RANK` is allowed only in the select list of a `SELECT` or `INSERT` statement or in the `ORDER BY` clause of the `SELECT` statement.

`PERCENT_RANK` can be in a view or a union. The `PERCENT_RANK` function cannot be used in a subquery, a `HAVING` clause, or in the select list of an `UPDATE` or `DELETE` statement. Only one rank analytical function is allowed per query.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

“Analytical functions” on page 104

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*
Alphabetical list of functions

**PERCENTILE_CONT function [Analytical]**

Function

Given a percentile, returns the value that corresponds to that percentile. Assumes a continuous distribution data model.

**Note** If you are simply trying to compute a percentile, use the NTILE function instead, with a value of 100.

| Syntax                  | PERCENTILE_CONT( expression1 ) WITHIN GROUP ( ORDER BY expression2 [ ASC | DESC ] ) |
|-------------------------|----------------------------------------------------------------------------------|
| Parameters              | expression1 A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, a “wrong argument for percentile” error is returned. If the argument value is less than 0 or greater than 1, a “data value out of range” error is returned. expression2 A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression. |

**Example**

The following example uses the PERCENTILE_CONT function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

<table>
<thead>
<tr>
<th>sales</th>
<th>region</th>
<th>dealer_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>900</td>
<td>Northeast</td>
<td>Boston</td>
</tr>
<tr>
<td>800</td>
<td>Northeast</td>
<td>Worcester</td>
</tr>
<tr>
<td>800</td>
<td>Northeast</td>
<td>Providence</td>
</tr>
<tr>
<td>700</td>
<td>Northeast</td>
<td>Lowell</td>
</tr>
<tr>
<td>540</td>
<td>Northeast</td>
<td>Natick</td>
</tr>
<tr>
<td>500</td>
<td>Northeast</td>
<td>New Haven</td>
</tr>
<tr>
<td>450</td>
<td>Northeast</td>
<td>Hartford</td>
</tr>
<tr>
<td>800</td>
<td>Northwest</td>
<td>SF</td>
</tr>
<tr>
<td>600</td>
<td>Northwest</td>
<td>Seattle</td>
</tr>
<tr>
<td>500</td>
<td>Northwest</td>
<td>Portland</td>
</tr>
<tr>
<td>400</td>
<td>Northwest</td>
<td>Dublin</td>
</tr>
<tr>
<td>500</td>
<td>South</td>
<td>Houston</td>
</tr>
<tr>
<td>400</td>
<td>South</td>
<td>Austin</td>
</tr>
<tr>
<td>300</td>
<td>South</td>
<td>Dallas</td>
</tr>
<tr>
<td>200</td>
<td>South</td>
<td>Dover</td>
</tr>
</tbody>
</table>

The following SELECT statement contains the PERCENTILE_CONT function:

```
SELECT region, PERCENTILE_CONT(0.1) WITHIN GROUP ( ORDER BY sales DESC )
```
FROM carSales GROUP BY region;

The result of the SELECT statement lists the 10th percentile value for car sales in a region:

<table>
<thead>
<tr>
<th>region</th>
<th>percentile_cont</th>
</tr>
</thead>
<tbody>
<tr>
<td>Northeast</td>
<td>840</td>
</tr>
<tr>
<td>Northwest</td>
<td>740</td>
</tr>
<tr>
<td>South</td>
<td>470</td>
</tr>
</tbody>
</table>

Usage

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function PERCENTILE_CONT takes a percentile value as the function argument, and operates on a group of data specified in the WITHIN GROUP clause, or operates on the entire data set. The function returns one value per group. If the GROUP BY column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its ORDER BY item specified in the WITHIN GROUP clause. The data type of the ORDER BY expression for PERCENTILE_CONT must be numeric.

PERCENTILE_CONT requires a WITHIN GROUP (ORDER BY) clause.

The ORDER BY clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. For the PERCENTILE_CONT function, the data type of this expression must be numeric. This ORDER BY clause is used only within the WITHIN GROUP clause and is not an ORDER BY for the SELECT.

The WITHIN GROUP clause distributes the query result into an ordered data set from which the function calculates a result. The WITHIN GROUP clause must contain a single sort item. If the WITHIN GROUP clause contains more or less than one sort item, an error is reported.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The PERCENTILE_CONT function is allowed in a subquery, a HAVING clause, a view, or a union. PERCENTILE_CONT can be used anywhere the simple nonanalytical aggregate functions are used. The PERCENTILE_CONT function ignores the NULL value in the data set.

Standards and compatibility

- **SQL92** vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

“Analytical functions” on page 104
“NTILE function [Analytical]” on page 212
PERCENTILE_DISC function [Analytical]

Function

Given a percentile, returns the value that corresponds to that percentile. Assumes a discrete distribution data model.

**Note** If you are simply trying to compute a percentile, use the NTILE function instead, with a value of 100.

Syntax

```
PERCENTILE_DISC ( expression1 )
WITHIN GROUP ( ORDER BY expression2 [ ASC | DESC ])
```

Parameters

- **expression1** A constant of numeric data type and range from 0 to 1 (inclusive). If the argument is NULL, then a “wrong argument for percentile” error is returned. If the argument value is less than 0 or greater than 1, then a “data value out of range” error is returned.

- **expression2** A sort specification that must be a single expression involving a column reference. Multiple expressions are not allowed and no rank analytical functions, set functions, or subqueries are allowed in this sort expression.

Example

The following example uses the PERCENTILE_DISC function to determine the 10th percentile value for car sales in a region.

The following data set is used in the example:

```
sales  region  dealer_name
900    Northeast  Boston
800    Northeast  Worcester
800    Northeast  Providence
700    Northeast  Lowell
540    Northeast  Natick
500    Northeast  New Haven
450    Northeast  Hartford
800    Northwest  SF
600    Northwest  Seattle
500    Northwest  Portland
400    Northwest  Dublin
500    South    Houston
400    South    Austin
300    South    Dallas
```
The following SELECT statement contains the PERCENTILE_DISC function:

```
SELECT region, PERCENTILE_DISC(0.1)
  WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

The result of the SELECT statement lists the 10th percentile value for car sales in a region:

<table>
<thead>
<tr>
<th>region</th>
<th>percentile_cont</th>
</tr>
</thead>
<tbody>
<tr>
<td>Northeast</td>
<td>900</td>
</tr>
<tr>
<td>Northwest</td>
<td>800</td>
</tr>
<tr>
<td>South</td>
<td>500</td>
</tr>
</tbody>
</table>

**Usage**

The inverse distribution analytical functions return a k-th percentile value, which can be used to help establish a threshold acceptance value for a set of data. The function PERCENTILE_DISC takes a percentile value as the function argument and operates on a group of data specified in the WITHIN GROUP clause or operates on the entire data set. The function returns one value per group. If the GROUP BY column from the query is not present, the result is a single row. The data type of the results is the same as the data type of its ORDER BY item specified in the WITHIN GROUP clause. PERCENTILE_DISC supports all data types that can be sorted in Sybase IQ.

PERCENTILE_DISC requires a WITHIN GROUP (ORDER BY) clause.

The ORDER BY clause, which must be present, specifies the expression on which the percentile function is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the WITHIN GROUP clause and is not an ORDER BY for the SELECT.

The WITHIN GROUP clause distributes the query result into an ordered data set from which the function calculates a result. The WITHIN GROUP clause must contain a single sort item. If the WITHIN GROUP clause contains more or less than one sort item, an error is reported.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

The PERCENTILE_DISC function is allowed in a subquery, a HAVING clause, a view, or a union. PERCENTILE_DISC can be used anywhere the simple nonanalytical aggregate functions are used. The PERCENTILE_DISC function ignores the NULL value in the data set.

**Standards and compatibility**

- **SQL92** Vendor extension
Alphabetical list of functions

- **Sybase**  Not supported by Adaptive Server Enterprise or SQL Anywhere.

See also

- “Analytical functions” on page 104
- “NTILE function [Analytical]” on page 212
- “PERCENTILE_CONT function [Analytical]” on page 220

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

---

**PI function [Numeric]**

Function

Returns the numeric value PI.

Syntax

```
PI ( )
```

Example

The following statement returns the value 3.141592653....

```
SELECT PI( ) FROM iq_dummy
```

Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  The PI() function is supported in Adaptive Server Enterprise, but PI(*) is not.

---

**POWER function [Numeric]**

Function

Calculates one number raised to the power of another.

Syntax

```
POWER ( numeric-expression1, numeric-expression2 )
```

Parameters

- `numeric-expression1`  The base.
- `numeric-expression2`  The exponent.

Example

The following statement returns the value 64:

```
SELECT POWER( 2, 6 ) FROM iq_dummy
```

Usage

Raises `numeric-expression1` to the power `numeric-expression2`.

Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  Compatible with Adaptive Server Enterprise.
PROPERTY function [System]

Function
Returns the value of the specified server-level property as a string.

Syntax
```sql
PROPERTY ({ property-id | property-name })
```

Note

Parameters
- **property-id** An integer that is the property-number of the server-level property. This number can be determined from the PROPERTY_NUMBER function. The `property-id` is commonly used when looping through a set of properties.
- **property-name** A string giving the name of the property. See “Properties available for the server” on page 121 for a list of server property names.

Example
The following statement returns the name of the current database server:
```sql
SELECT PROPERTY('Name') FROM iq_dummy
```

Usage
Each property has both a number and a name, but the number is subject to change between versions, and should not be used as a reliable identifier for a given property.

Standards and compatibility
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

See also
“Properties available for the server” on page 121

PROPERTY_DESCRIPTION function [System]

Function
Returns a description of a property.

Syntax
```sql
PROPERTY_DESCRIPTION ({ property-id | property-name })
```

Note

Parameters
- **property-id** An integer that is the property number of the property. This number can be determined from the PROPERTY_NUMBER function. The `property-id` is commonly used when looping through a set of properties.
**PROPERTY_NAME function [System]**

**Function**

Returns the name of the property with the supplied property number.

**Syntax**

```
PROPERTY_NAME ( property-id )
```

**Parameters**

- `property-id` The property number of the property.

**Example**

The following statement returns the property associated with property number 126. The actual property to which this refers changes from version to version.

```
SELECT PROPERTY_NAME ( 126 ) FROM iq_dummy
```

**Standards and compatibility**

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise.

**See also**

- “Connection properties” on page 121
- “Properties available for the server” on page 121
- “Properties available for each database” on page 122
PROPERTY_NUMBER function [System]

Function Returns the property number of the property with the supplied property name.

Syntax

    PROPERTY_NUMBER ( property-name )

Parameters

    property-name A property name. For property names, see the lists in “Connection properties” on page 121, “Properties available for the server” on page 121, and “Properties available for each database” on page 122.

Example

The following statement returns an integer value. The actual value changes from version to version.

    SELECT PROPERTY_NUMBER( 'PAGESIZE' ) FROM iq_dummy

Standards and compatibility

- SQL92 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise

See also

“Connection properties” on page 121
“Properties available for the server” on page 121
“Properties available for each database” on page 122

QUARTER function [Date and time]

Function Returns a number indicating the quarter of the year from the supplied date expression.

Syntax

    QUARTER ( date-expression )

Parameters

    date-expression A date.

Example

With the DATE_ORDER option set to the default of ymd, the following statement returns the value 2:

    SELECT QUARTER ( '1987/05/02' ) FROM iq_dummy

Usage

Table 4-26 lists the dates in the quarters of the year.
Table 4-26: Values of quarter of the year

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Period (inclusive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>January 1 to March 31</td>
</tr>
<tr>
<td>2</td>
<td>April 1 to June 30</td>
</tr>
<tr>
<td>3</td>
<td>July 1 to September 30</td>
</tr>
<tr>
<td>4</td>
<td>October 1 to December 31</td>
</tr>
</tbody>
</table>

Standards and compatibility

- **SQL92**: Vendor extension
- **Sybase**: Not supported by Adaptive Server Enterprise

See also

DATE_ORDER option in Reference: Statements and Options

---

**RADIANS function [Numeric]**

**Function**: Converts a number from degrees to radians.

**Syntax**: `RADIANS ( numeric-expression )`

**Parameters**

- **numeric-expression**: A number, in degrees. This angle is converted to radians.

**Example**: The following statement returns a value of approximately 0.5236:

```
SELECT RADIANS( 30 ) FROM iq_dummy
```

Standards and compatibility

- **SQL92**: Vendor extension
- **Sybase**: Compatible with Adaptive Server Enterprise

---

**RAND function [Numeric]**

**Function**: Returns a DOUBLE precision, random number x, where 0 <= x <1, with an optional seed.

**Syntax**: `RAND([ integer-expression ])`

**Parameters**

- **integer-expression**: The optional seed used to create a random number. This argument allows you to create repeatable random number sequences.

If RAND is called with a FROM clause and an argument in a query containing only tables in IQ stores, the function returns an arbitrary but repeatable value.
When no argument is called, RAND is a non-deterministic function. Successive calls to RAND might return different values. The query optimizer does not cache the results of the RAND function.

**Note** The values returned by RAND vary depending on whether you use a FROM clause or not and whether the referenced table was created in SYSTEM or in an IQ store.

### Examples

The following statement returns a 5% sampling of a table:

```
SELECT AVG(table1.number_of_cars),
       AVG(table1.number_of_tvs)
FROM table1 WHERE
       RAND(ROWID(table1)) < .05 and table1.income < 50000;
```

The following statement returns a value of approximately 941392926249216914:

```
SELECT RAND( 4 ) FROM iq_dummy
```

### Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

### RANK function [Analytical]

**Function** Ranks items in a group.

**Syntax**

```
RANK () OVER ( ORDER BY expression [ ASC | DESC ] )
```

**Parameters**

- **expression** A sort specification that can be any valid expression involving a column reference, aggregates, or expressions invoking these items.

**Example**

The following statement illustrates the use of the RANK function:

```
SELECT s_suppkey, SUM(s_acctBal) AS sum_acctBal,
       RANK() OVER ( ORDER BY SUM(s_acctBal) DESC )
       AS rank_all
FROM supplier GROUP BY s_suppkey;
```

<table>
<thead>
<tr>
<th>s_suppkey</th>
<th>sum_acctBal</th>
<th>rank_all</th>
</tr>
</thead>
<tbody>
<tr>
<td>supplier#011</td>
<td>200000</td>
<td>1</td>
</tr>
<tr>
<td>supplier#002</td>
<td>200000</td>
<td>1</td>
</tr>
<tr>
<td>supplier#013</td>
<td>123000</td>
<td>3</td>
</tr>
<tr>
<td>supplier#004</td>
<td>110000</td>
<td>4</td>
</tr>
<tr>
<td>supplier#035</td>
<td>110000</td>
<td>4</td>
</tr>
<tr>
<td>supplier#006</td>
<td>50000</td>
<td>6</td>
</tr>
<tr>
<td>supplier#021</td>
<td>10000</td>
<td>7</td>
</tr>
</tbody>
</table>
Usage

RANK is a rank analytical function. The rank of row R is defined as the number of rows that precede R and are not peers of R. If two or more rows are not distinct within the groups specified in the OVER clause or distinct over the entire result set, then there are one or more gaps in the sequential rank numbering. The difference between RANK and DENSE_RANK is that DENSE_RANK leaves no gap in the ranking sequence when there is a tie. RANK leaves a gap when there is a tie.

RANK requires an OVER (ORDER BY) clause. The ORDER BY clause specifies the parameter on which ranking is performed and the order in which the rows are sorted in each group. This ORDER BY clause is used only within the OVER clause and is not an ORDER BY for the SELECT. No aggregation functions in the rank query are allowed to specify DISTINCT.

The OVER clause indicates that the function operates on a query result set. The result set is the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses have all been evaluated. The OVER clause defines the data set of the rows to include in the computation of the rank analytical function.

The ASC or DESC parameter specifies the ordering sequence ascending or descending. Ascending order is the default.

RANK is allowed only in the select list of a SELECT or INSERT statement or in the ORDER BY clause of the SELECT statement. RANK can be in a view or a union. The RANK function cannot be used in a subquery, a HAVING clause, or in the select list of an UPDATE or DELETE statement. Only one rank analytical function is allowed per query.

Standards and compatibility

- SQL92 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise or SQL Anywhere

See also

“Analytical functions” on page 104
“DENSE_RANK function [Analytical]” on page 163

REGR_AVGX function [Aggregate]

Function

Computes the average of the independent variable of the regression line.

Syntax 1

```
REGR_AVGX (dependent-expression, independent-expression)
```

Syntax 2

```
REGR_AVGX (dependent-expression, independent-expression)
OVER (window-spec)
```
Parameters

- **dependent-expression**: The variable that is affected by the independent variable.
- **independent-expression**: The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then `REGR_AVGX` returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where x represents the independent-expression:

\[
\text{AVG}(x)
\]

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at [SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions](#).

**Note** ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of `window-spec` either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example calculates the average of the dependent variable, employee age:

```sql
SELECT REGR_AVGX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

Standards and compatibility

- **SQL2008**: SQL foundation feature (T621) outside of core SQL
- **Sybase**: Compatible with SQL Anywhere
REGR_AVGY function [Aggregate]

Function

Computes the average of the dependent variable of the regression line.

Syntax 1

REGR_AVGY(dependent-expression, independent-expression)

Syntax 2

REGR_AVGY(dependent-expression, independent-expression)

OVER (window-spec)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

- dependent-expression: The variable that is affected by the independent variable.
- independent-expression: The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_AVGY returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the dependent-expression:

\[
\text{AVG}(y)
\]

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions.

Note

ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example calculates the average of the independent variable, employee salary. This function returns the value 49988.6232:

```sql
SELECT REGR_AVGY( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) )
```
FROM Employees;

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

**REGR_COUNT function [Aggregate]**

Function

Returns an integer that represents the number of non-NULL number pairs used to fit the regression line.

Syntax 1

```sql
REGR_COUNT(dependent-expression, independent-expression)
```

Syntax 2

```sql
REGR_COUNT(dependent-expression, independent-expression)
OVER (window-spec)
```

*window-spec*: See Syntax 2 instructions in the Usage section, below.

Parameters

- **dependent-expression** The variable that is affected by the independent variable.
- **independent-expression** The variable that influences the outcome.

Usage

This function returns an UNSIGNED BIGINT as the result.

**Note** ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of *window-spec* either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns a value that indicates the number of non-NULL pairs that were used to fit the regression line. This function returns the value 75:

```sql
SELECT REGR_COUNT( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQ.
- **Sybase** Compatible with SQL Anywhere
REGR_INTERCEPT function [Aggregate]

Function

Computes the y-intercept of the linear regression line that best fits the dependent and independent variables.

Syntax 1

REGR_INTERCEPT(dependent-expression, independent-expression)

Syntax 2

REGR_INTERCEPT(dependent-expression, independent-expression)

OVER (window-spec)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, REGR_INTERCEPT returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the independent-expression:

AVG(y) - REGR_SLOPE(y, x) * AVG(x)

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns the value 1874.580568517603:

SELECT REGR_INTERCEPT( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;

Standards and compatibility

• SQL2008 SQL foundation feature (T621) outside of core SQL
• Sybase Compatible with SQL Anywhere

REGR_R2 function [Aggregate]

Function

Computes the coefficient of determination (also referred to as R-squared or the goodness-of-fit statistic) for the regression line.

Syntax 1

REGR_R2(dependent-expression, independent-expression)

Syntax 2

REGR_R2(dependent-expression, independent-expression)
OVER (window-spec)

window-spec: See Syntax 2 instructions in the Usage section below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_R2 returns NULL.

REGR_R2 is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. IQ then applies the following algorithm:

• REGR_R2 calculates VAR_POP(x) and returns NULL if VAR_POP(x) = 0; otherwise, it calculates VAR_POP(y) and returns the value 1 if VAR_POP(y) = 0.
• If neither VAR_POP(x) or VAR_POP(y) is zero, the return value is POWER(CORR(y,x),2)

where y represents the dependent-expression and x represents the independent-expression.
Alphabetical list of functions

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns the value 0.19379959710325653:

```
SELECT REGR_R2( Salary, ( YEAR( NOW() ) - YEAR(BirthDate) ) )
FROM Employees;
```

Standards and compatibility

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

### REGR_SLOPE function [Aggregate]

**Function** Computes the slope of the linear regression line, fitted to non-NULL pairs.

**Syntax 1**

```
REGR_SLOPE(dependent-expression, independent-expression)
```

**Syntax 2**

```
REGR_SLOPE(dependent-expression, independent-expression)
OVER (window-spec)
```

*window-spec*: See Syntax 2 instructions in the Usage section below.

**Parameters**

- **dependent-expression** The variable that is affected by the independent variable.
- **independent-expression** The variable that influences the outcome.

**Usage**

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_SLOPE returns NULL.
REGR_SLOPE is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the independent-expression:

\[
\frac{\text{COV AR POP}(x, y)}{\text{VAR POP}(y)}
\]

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions.

**Note** ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

**Example**

The following example returns the value 935.3429749445614:

```sql
SELECT REGR_SLOPE(Salary, (YEAR(NOW()) - YEAR(BirthDate)))
FROM Employees;
```

**Standards and compatibility**

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

### REGR_SXX function [Aggregate]

**Function**
Computes the slope of the linear regression line, fitted to non-NULL pairs.

**Syntax 1**

```
REGR_SXX(dependent-expression, independent-expression)
```

**Syntax 2**

```
REGR_SXX(dependent-expression, independent-expression)
OVER (window-spec)
```

*window-spec*: See Syntax 2 instructions in the Usage section below.
Alphabetical list of functions

<table>
<thead>
<tr>
<th>Parameters</th>
<th>dependent-expression</th>
<th>The variable that is affected by the independent variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>independent-expression</td>
<td>The variable that influences the outcome.</td>
</tr>
</tbody>
</table>

**Usage**

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then **REGR_SXX** returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the independent-expression:

\[
\text{REGR_COUNT}(y, x) \times \text{VAR_POP}(x)
\]

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at [SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions](#).

**Note** ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

**Syntax 2** represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

**Example**

The following example returns the value 5916.4800000000105:

```sql
SELECT REGR_SXX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

**Standards and compatibility**

- **SQL2008** SQL foundation feature (T621) outside of core SQL
- **Sybase** Compatible with SQL Anywhere

**REGR_SXY function [Aggregate]**

**Function**

Returns the sum of products of the dependent and independent variables. Use **REGR_SXY** to evaluate the statistical validity of a regression model.
Syntax 1

REGR_SXY(dependent-expression, independent-expression)

Syntax 2

REGR_SXY(dependent-expression, independent-expression)

OVER (window-spec)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression  The variable that is affected by the independent variable.

independent-expression  The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then it returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is made, where y represents the dependent-expression and x represents the independent-expression:

\[
REGR_COUNT(x, y) \times COV AR\_POP(x, y)
\]

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions.

Note  ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns the value 5533938.004400015.

```sql
SELECT REGR_SXY( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

Standards and compatibility

- **SQL2008**: SQL foundation feature (T621) outside of core SQL
- **Sybase**: Compatible with SQL Anywhere
REGR_SYY function [Aggregate]

Function

Returns values that can evaluate the statistical validity of a regression model.

Syntax 1

REGR_SYY(dependent-expression, independent-expression)

Syntax 2

REGR_SYY(dependent-expression, independent-expression)

OVER (window-spec)

window-spec: See Syntax 2 instructions in the Usage section, below.

Parameters

dependent-expression The variable that is affected by the independent variable.

independent-expression The variable that influences the outcome.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating-point, and returns a DOUBLE as the result. If applied to an empty set, then REGR_SYY returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the dependent-expression and x represents the independent-expression:

REGR_COUNT(x, y) * VAR_POP(y)

See “Mathematical formulas for the aggregate functions” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Usage > Querying and Modifying Data > OLAP support > Window functions in SQL Anywhere > Row numbering functions.

Note ROLLUP and CUBE are not supported in the GROUP BY clause with Syntax 1. DISTINCT is not supported.

Syntax 2 represents usage as a window function in a SELECT statement. As such, you can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns the value 26, 708, 672,843.3002:

SELECT REGR_SYY( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;

Note
Standards and compatibility

- **SQL2008**: SQL foundation feature (T621) outside of core SQL
- **Sybase**: Compatible with SQL Anywhere

**REMAINDER function [Numeric]**

**Function**

Returns the remainder when one whole number is divided by another.

**Syntax**

```
REMAINDER ( dividend, divisor )
```

**Parameters**

- `dividend`: The dividend, or numerator of the division.
- `divisor`: The divisor, or denominator of the division.

**Example**

The following statement returns the value 2:

```sql
SELECT REMAINDER( 5, 3 ) FROM iq_dummy
```

**Usage**

`REMAINDER` is the same as the `MOD` function.

**Standards and compatibility**

- **SQL92**: Vendor extension
- **Sybase**: Not supported in Adaptive Server Enterprise. The `%` (modulo) operator and the division operator can be used to produce a remainder.

**See also**

“MOD function [Numeric]” on page 205

**REPEAT function [String]**

**Function**

Concatenates a string a specified number of times.

**Syntax**

```
REPEAT ( string-expression, integer-expression )
```

**Parameters**

- `string-expression`: The string to be repeated.
- `integer-expression`: The number of times the string is to be repeated. If `integer-expression` is negative, an empty string is returned.

**Note**

The result data type of a `REPEAT` function is a `LONG VARCHAR`. If you use `REPEAT` in a `SELECT INTO` statement, you must have a Large Objects Management option license or use `CAST` and set `REPEAT` to the correct data type and size.

See “REPLACE function [String]” for more information.

**Example**

The following statement returns the value “repeatrepeatrepeat:”

```sql
SELECT REPEAT( 'repeat', 3 ) FROM iq_dummy
```
Alphabetical list of functions

Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  Not supported in Adaptive Server Enterprise, but REPLICATE provides the same capabilities

See also

“REPLICATE function [String]” on page 243

REPLACE function [String]

**Function**
Replaces all occurrences of a substring with another substring.

**Syntax**

```
REPLACE (original-string, search-string, replace-string)
```

**Parameters**

If any argument is NULL, the function returns NULL.

- **original-string**  The string to be searched. This string can be any length.
- **search-string**  The string to be searched for and replaced with **replace-string**. This string is limited to 255 bytes. If **search-string** is an empty string, the original string is returned unchanged.
- **replace-string**  The replacement string, which replaces **search-string**. This can be any length. If **replace-string** is an empty string, all occurrences of **search-string** are deleted.

If you need to control the width of the resulting column when **replace-string** is wider than **search-string**, use the **CAST** function. For example,

```
CREATE TABLE aa(a CHAR(5));
INSERT INTO aa VALUES('CCCCC');
COMMIT;
SELECT a, CAST(REPLACE(a,'C','ZZ') AS CHAR(5)) FROM aa;
```

**Examples**

The following statement returns the value “xx.def.xx.ghi:”

```
SELECT REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' ) FROM iq_dummy
```

The following statement generates a result set containing ALTER PROCEDURE statements which, when executed, repair stored procedures that reference a table that has been renamed. (To be useful, the table name must be unique.)

```
SELECT REPLACE(
    replace(proc_defn,'OldTableName','NewTableName'),
    'create procedure',
    'alter procedure')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%'
```

Use a separator other than the comma for the LIST function:
SELECT REPLACE( list( table_id ), ',', '--' )
FROM SYS.ISYSTAB
WHERE table_id <= 5

Usage

The result data type of a REPLACE function is a LONG VARCHAR. If you use REPLACE in a SELECT INTO statement, you must have a Large Objects Management option license, or use CAST and set REPLACE to the correct data type and size.

There are two ways to work around this issue:

- Declare a local temporary table, then perform an INSERT:

  DECLARE local temporary table #mytable
  (name_column char(10)) on commit preserve rows;
  INSERT INTO #mytable SELECT REPLACE(name,'0','1')
  FROM dummy_table01;

- Use CAST:

  SELECT CAST(replace(name, '0', '1') AS Char(10))
  into #mytable from dummy_table01;

Standards and compatibility

- SQL92  Vendor extension
- Sybase  Compatible with Adaptive Server Enterprise

See also

“SUBSTRING function [String]” on page 265

**REPLICATE function [String]**

**Function**

Concatenates a string a specified number of times.

**Syntax**

```
REPLICATE ( string-expression, integer-expression )
```

**Parameters**

- `string-expression`  The string to be repeated.
- `integer-expression`  The number of times the string is to be repeated.

**Example**

The following statement returns the value “repeatrepeatrepeat:”

```
SELECT REPLICATE( 'repeat', 3 ) FROM iq_dummy
```
REPLICATE is the same as the REPEAT function.

**Usage**

**Note** The result data type of a REPLICATE function is a LONG VARCHAR. If you use REPLICATE in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set REPLICATE to the correct data type and size.

See “REPLACE function [String]” on page 242 for more information.

**Standards and compatibility**

- SQL92   Vendor extension
- Sybase   Compatible with Adaptive Server Enterprise

**See also**

“REPEAT function [String]” on page 241

### REVERSE function [String]

**Function**

Takes one argument as an input of type BINARY or STRING and returns the specified string with characters listed in reverse order.

**Syntax**

```
REVERSE ( expression | uchar_expr )
```

**Parameters**

- **expression** is a character or binary-type column name, variable, or constant expression of CHAR, VARCHAR, NCHAR, NVARCHAR, BINARY, or VARBINARY type.

**Example 1**

```
select reverse("abcd")
----
dcba
```

**Example 2**

```
select reverse(0x12345000)
----------
0x00503412
```

**Usage**

- REVERSE, a string function, returns the reverse of expression.
- If expression is NULL, reverse returns NULL.
- Surrogate pairs are treated as indivisible and are not reversed.

**Permissions**

Any user can execute REVERSE.

**Standards and compatibility**

ANSI SQL – Compliance level: Transact-SQL extension

**See also**

Functions “LOWER function [String]” on page 200 and “UPPER function [String]” on page 329.
For general information about string functions, see “String functions” on page 115.

**RIGHT function [String]**

**Function**
Returns the rightmost characters of a string.

**Syntax**
```
RIGHT ( string-expression, numeric-expression )
```

**Parameters**
- **string-expression** The string to be left-truncated.
- **numeric-expression** The number of characters at the end of the string to return.

**Example**
The following statement returns the value “olate:”
```
SELECT RIGHT( 'chocolate', 5 ) FROM iq_dummy
```

**Usage**
If the string contains multibyte characters, and the proper collation is being used, the number of bytes returned might be greater than the specified number of characters.

**Note**
The result data type of a RIGHT function is a LONG VARCHAR. If you use RIGHT in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set RIGHT to the correct data type and size.

See “REPLACE function [String]” on page 242 for more information.

**Standards and compatibility**
- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

**See also**
Chapter 11, “International Languages and Character Sets” in the System Administration Guide: Volume 1

**ROUND function [Numeric]**

**Function**
Rounds the numeric-expression to the specified integer-expression number of places after the decimal point.

**Syntax**
```
ROUND ( numeric-expression, integer-expression )
```

**Parameters**
- **numeric-expression** The number, passed to the function, to be rounded.
**Alphabetical list of functions**

**integer-expression**  A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

**Examples**

The following statement returns the value 123.200:

```
SELECT ROUND( 123.234, 1 ) FROM iq_dummy
```

Additional results of the ROUND function are shown in the following table:

<table>
<thead>
<tr>
<th>Value</th>
<th>ROUND (Value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>123.4567</td>
<td>round (a,n,4)</td>
</tr>
<tr>
<td>123.4570</td>
<td>round (a,n,3)</td>
</tr>
<tr>
<td>123.4600</td>
<td>round (a,n,2)</td>
</tr>
<tr>
<td>123.5000</td>
<td>round (a,n,1)</td>
</tr>
<tr>
<td>123.0000</td>
<td>round (a,n,0)</td>
</tr>
<tr>
<td>120.0000</td>
<td>round (a,n,-1)</td>
</tr>
<tr>
<td>100.0000</td>
<td>round (a,n,-2)</td>
</tr>
<tr>
<td>0.0000</td>
<td>round(a,n,-3)</td>
</tr>
</tbody>
</table>

**Standards and compatibility**

- **SQL92**  Vendor extension
- **Sybase**  Compatible with Adaptive Server Enterprise

**See also**

“TRUNCNUM function [Numeric]” on page 268

**ROWID function [Miscellaneous]**

**Function**  Returns the internal row ID value for each row of the table.

**Syntax**  

```
ROWID ( table-name ) FROM table-name
```

**Parameters**

- **table-name**  The name of the table. Specify the name of the table within the parentheses with either no quotes or with double quotes.

**Examples**

The following statement returns the row ID values 1 through 10:

```
SELECT ROWID( "PRODUCTS" ) FROM PRODUCTS
```
The following statement returns the product ID and row ID value of all rows with a product ID value less than 400:

```sql
SELECT PRODUCTS.ID, ROWID ( PRODUCTS )
FROM PRODUCTS
WHERE PRODUCTS.ID < 400
```

<table>
<thead>
<tr>
<th>ID</th>
<th>rowid(Products)</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>1</td>
</tr>
<tr>
<td>301</td>
<td>2</td>
</tr>
<tr>
<td>302</td>
<td>3</td>
</tr>
</tbody>
</table>

The following statement deletes all rows with row ID values greater than 50:

```sql
DELETE FROM PRODUCTS
WHERE ROWID ( PRODUCTS ) > 50
```

Usage

You can use the `ROWID` function in conjunction with other clauses to manipulate specific rows of the table.

You must specify the `FROM table-name` clause.

A limitation of the `ROWID` function is that it cannot use a join index of that table, eliminating any performance benefits that would normally use that join index.

Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  Not supported by Adaptive Server Enterprise

**RTRIM function [String]**

Function

Returns a string with trailing blanks removed.

Syntax

```sql
RTRIM ( string-expression )
```
Alphabetical list of functions

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>string-expression</strong></td>
<td>The string to be trimmed.</td>
</tr>
</tbody>
</table>

**Note** The result data type of an RTRIM function is a LONG VARCHAR. If you use RTRIM in a SELECT INTO statement, you must have a Large Objects Management option license or use CAST and set RTRIM to the correct data type and size.

See “REPLACE function [String]” on page 242 for more information.

### Example

The following statement returns the string “Test Message” with all trailing blanks removed.

```sql
SELECT RTRIM( 'Test Message     ' ) FROM iq_dummy
```

### Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

### See also

“LTRIM function [String]” on page 201

---

### SECOND function [Date and time]

**Function** Returns a number from 0 to 59 corresponding to the second component of the given date/time value.

**Syntax**

```sql
SECOND ( datetime-expression )
```

**Parameters**

- **datetime-expression** The date/time value.

**Example**

The following statement returns the value 5:

```sql
SELECT SECOND( '1998-07-13 08:21:05' ) FROM iq_dummy
```

### Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Compatible with Adaptive Server Enterprise

---

### SECONDS function [Date and time]

**Function** Returns the number of seconds since an arbitrary starting date and time, the number of seconds between two times, or adds an integer amount of seconds to a time.

**Syntax**

```sql
SECONDS ( datetime-expression
| datetime-expression, datetime-expression
| datetime-expression, integer-expression )
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>datetime-expression</code></td>
<td>A date and time.</td>
</tr>
<tr>
<td><code>integer-expression</code></td>
<td>The number of seconds to be added to the <code>datetime-expression</code>. If <code>integer-expression</code> is negative, the appropriate number of minutes are subtracted from the date/time value. If you supply an integer expression, the <code>datetime-expression</code> must be explicitly cast as a datetime data type. For information on casting data types, see “CAST function [Data type conversion]” on page 133.</td>
</tr>
</tbody>
</table>

### Examples

The following statement returns the value 3600:

```sql
SELECT ( SECONDS( '1998-07-13 06:07:12' ) - SECONDS( '1998-07-13 05:07:12' ) ) FROM iq_dummy
```

The following statement returns the value 14400, to signify the difference between the two times:

```sql
```

The following statement returns the datetime value 1999-05-12 21:05:12.000:

```sql
SELECT SECONDS( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5 ) FROM iq_dummy
```

### Usage

The second syntax returns the number of whole seconds from the first date/time to the second date/time. The number might be negative.

### Standards and compatibility

- **SQL92**  
  Vendor extension

- **Sybase**  
  Not supported by Adaptive Server Enterprise

### SIGN function [Numeric]

**Function**

Returns the sign of a number.

**Syntax**

```sql
SIGN ( numeric-expression )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>numeric-expression</code></td>
<td>The number for which the sign is to be returned.</td>
</tr>
</tbody>
</table>

**Example**

The following statement returns the value -1:

```sql
SELECT SIGN( -550 ) FROM iq_dummy
```

**Usage**

For negative numbers, `SIGN` returns -1.

For zero, `SIGN` returns 0.

For positive numbers, `SIGN` returns 1.
SIMILAR function [String]

Function
Returns an integer between 0 and 100 representing the similarity between two strings.

Syntax
SIMILAR( string-expression1, string-expression2 )

Parameters
string-expression1 The first string to be compared.
string-expression2 The second string to be compared.

Example
The following statement returns the value 75:
SELECT SIMILAR( 'toast', 'coast' ) FROM iq_dummy

This signifies that the two values are 75% similar.

Usage
The function returns an integer between 0 and 100 representing the percentage of characters matched between the two strings. A value of 100 indicates that the two strings are identical.

This function can be used to correct a list of names (such as customers). Some customers might have been added to the list more than once with slightly different names. Join the table to itself and produce a report of all similarities greater than 90 percent but less than 100 percent.

SIN function [Numeric]

Function
Returns the sine of a number, expressed in radians.

Syntax
SIN( numeric-expression )

Parameters
numeric-expression The angle, in radians.

Example
The following statement returns the value 0.496880:
SELECT SIN( 0.52 ) FROM iq_dummy

Standards and compatibility
• SQL92 Vendor extension
• Sybase Compatible with Adaptive Server Enterprise

Standards and compatibility
• SQL92 Vendor extension
• Sybase Not supported by Adaptive Server Enterprise
SORTKEY function [String]

Function
Generates values that can be used to sort character strings based on alternate collation rules.

Syntax
SORTKEY ( string-expression
[, { collation-id
| collation-name [(collation-tailoring-string)] } ]
)

Parameters

- **string-expression** The string expression must contain characters that are encoded in the character set of the database and must be STRING data type.

  If string-expression is NULL, the SORTKEY function returns a null value. An empty string has a different sort-order value than a null string from a database column.

  There is no limit on the length of the input string that the SORTKEY function can handle. The result of SORTKEY is always limited to 1024 bytes and is VARBINARY data type. If the actual results exceed 1024 bytes, the result contains only the first 1024 bytes.

- **collation-name** A string or character variable that specifies the name of the sort order to use. You can also specify the alias char_collation, or, equivalently, db_collation, to generate sort-keys as used by the CHAR collation in use by the database.

  Similarly, you can specify the alias NCHAR_COLLATION to generate sort-keys as used by the NCHAR collation in use by the database. However, Sybase IQ does not support NCHAR_COLLATION for IQ-specific objects. NCHAR_COLLATION is supported for SQL Anywhere objects on an IQ server.

- **collation-id** A variable, integer constant, or string that specifies the ID number of the sort order to use. This parameter applies only to Adaptive Server Enterprise collations, which can be referred to by their corresponding collation ID.
If you do not specify a collation name or collation ID, the default is Default Unicode multilingual.

Valid collations are as follows:

- To see collations that are supported by SQL Anywhere, listed by label, execute `dbinit -l`.
- The Adaptive Server Enterprise collations are listed in the table below.

<table>
<thead>
<tr>
<th>Description</th>
<th>Collation name</th>
<th>Collation ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Unicode multilingual</td>
<td>default</td>
<td>0</td>
</tr>
<tr>
<td>CP 850 Alternative: no accent</td>
<td>altnoacc</td>
<td>39</td>
</tr>
<tr>
<td>CP 850 Alternative: lowercase first</td>
<td>alldict</td>
<td>45</td>
</tr>
<tr>
<td>CP 850 Western European: no case, preference</td>
<td>altnocsp</td>
<td>46</td>
</tr>
<tr>
<td>CP 850 Scandinavian dictionary</td>
<td>scandict</td>
<td>47</td>
</tr>
<tr>
<td>CP 850 Scandinavian: no case, preference</td>
<td>scannocp</td>
<td>48</td>
</tr>
<tr>
<td>GB Pinyin</td>
<td>gbpinyin</td>
<td>n/a</td>
</tr>
<tr>
<td>Binary sort</td>
<td>binary</td>
<td>50</td>
</tr>
<tr>
<td>Latin-1 English, French, German dictionary</td>
<td>dict</td>
<td>51</td>
</tr>
<tr>
<td>Latin-1 English, French, German no case</td>
<td>nocase</td>
<td>52</td>
</tr>
<tr>
<td>Latin-1 English, French, German no case, preference</td>
<td>nocasep</td>
<td>53</td>
</tr>
<tr>
<td>Latin-1 English, French, German no accent</td>
<td>noaccent</td>
<td>54</td>
</tr>
<tr>
<td>Latin-1 Spanish dictionary</td>
<td>espdict</td>
<td>55</td>
</tr>
<tr>
<td>Latin-1 Spanish no case</td>
<td>espnocsp</td>
<td>56</td>
</tr>
<tr>
<td>Latin-1 Spanish no accent</td>
<td>espnoac</td>
<td>57</td>
</tr>
<tr>
<td>ISO 8859-5 Russian dictionary</td>
<td>rusdict</td>
<td>58</td>
</tr>
<tr>
<td>ISO 8859-5 Russian no case</td>
<td>rusnocsp</td>
<td>59</td>
</tr>
<tr>
<td>ISO 8859-5 Cyrillic dictionary</td>
<td>cyrdict</td>
<td>63</td>
</tr>
<tr>
<td>ISO 8859-5 Cyrillic no case</td>
<td>cymnocsp</td>
<td>64</td>
</tr>
<tr>
<td>ISO 8859-7 Greek dictionary</td>
<td>elldict</td>
<td>65</td>
</tr>
<tr>
<td>ISO 8859-2 Hungarian dictionary</td>
<td>hundict</td>
<td>69</td>
</tr>
<tr>
<td>ISO 8859-2 Hungarian no accents</td>
<td>hunnoac</td>
<td>70</td>
</tr>
<tr>
<td>ISO 8859-2 Hungarian no case</td>
<td>hunnocsp</td>
<td>71</td>
</tr>
<tr>
<td>ISO 8859-5 Turkish dictionary</td>
<td>turdict</td>
<td>72</td>
</tr>
<tr>
<td>ISO 8859-5 Turkish no accents</td>
<td>turnoac</td>
<td>73</td>
</tr>
<tr>
<td>ISO 8859-5 Turkish no case</td>
<td>turnocsp</td>
<td>74</td>
</tr>
<tr>
<td>CP 874 (TIS 620) Royal Thai dictionary</td>
<td>thaidict</td>
<td>1</td>
</tr>
</tbody>
</table>
(Optional) Specify collation tailoring options (collation-tailoring-string) for additional control over sorting and comparison of characters. These options take the form of keyword=value pairs assembled in parentheses, following the collation name. For example,

```
'UCA(locale=es;case=LowerFirst;accent=respect)'
```

The syntax for specifying these options is identical to the COLLATION clause of the CREATE DATABASE statement. See `collation-tailoring-string` in “CREATE DATABASE statement” in Chapter 1, “SQL Statements” of Reference: Statements and Options.

**Note** All of the collation tailoring options are supported for SQL Anywhere databases, when specifying the Unicode Collation Algorithm (UCA) collation. For all other collations, only case sensitivity tailoring is supported.

### Example

The following statement queries the Employees table and returns the FirstName and Surname of all employees, sorted by the sort-key values for the Surname column using the dict collation (Latin-1, English, French, German dictionary):

```
SELECT Surname, GivenName FROM Employees ORDER BY
SORTKEY( Surname, 'dict' );
```

### Usage

The SORTKEY function generates values that can be used to order results based on predefined sort order behavior. This allows you to work with character sort order behaviors that may not be available from the database collation. The returned value is a binary value that contains coded sort order information for the input string that is retained from the SORTKEY function.
For example, you can store the values returned by the \texttt{SORTKEY} function in a column with the source character string. The following \texttt{SELECT} statement retrieves data from table \texttt{T1} in the sorted order of \texttt{c1} according to the Thai dictionary:

\begin{verbatim}
SELECT rid, c1 from T1 ORDER BY SORTKEY(c1)
\end{verbatim}

You instead store the value returned by \texttt{SORTKEY} in a column with the source character string. To retrieve the character data in the required order, the \texttt{SELECT} statement needs to include only an \texttt{ORDER BY} clause on the column that contains the results of running the \texttt{SORTKEY} function:

\begin{verbatim}
UPDATE T1 SET shadowc1=SORTKEY(c1) FROM T1;
SELECT rid, c1 FROM T1 ORDER BY shadowc1
\end{verbatim}

The \texttt{SORTKEY} function guarantees that the values it returns for a given set of sort order criteria work for the binary comparisons that are performed on \texttt{VARBINARY} data types.

Generating sort-keys for queries can be expensive. As an alternative for frequently requested sort-keys, consider creating a computed column to hold the sort-key values, and then referencing that column in the \texttt{ORDER BY} clause of the query.

With respect to collation tailoring, full sensitivity is generally the intent when creating sort-keys, so when you specify a non-UCA collation, the default tailoring applied is equivalent to case=Respect. For example, the following two statements are equivalent:

\begin{verbatim}
SELECT SORTKEY( 'abc', '1252LATIN1' );
SELECT SORTKEY( 'abc', '1252LATIN1(case=Respect)' );
\end{verbatim}

When specifying a non-UCA collation, by default, collation tailorings are accent and case-sensitive. However, for non-UCA collations, you can override only the case sensitivity using a collation tailoring. For example:

\begin{verbatim}
SELECT SORTKEY( 'abc', '1252LATIN1(case=LowerFirst)' );
\end{verbatim}

If the database was created without specifying tailoring options, the following two clauses may generate different sort orders, even if the database collation name is specified for the \texttt{SORTKEY} function:

\begin{verbatim}
ORDER BY string-expression
ORDER BY SORTKEY( string-expression, database-collation-name )
\end{verbatim}
Different sort orders may be generated, because the default tailoring settings used for database creation and for the SORTKEY function are different. To get the same behavior from SORTKEY as for the database collation, either provide a tailoring syntax for `collation-tailoring-string` that matches the settings for the database collation, or specify `db_collation` for `collation-name`. For example:

```sql
SORTKEY( expression, 'db_collation' )
```

For information on using collation tailoring and the SORTKEY function with a SQL Anywhere database, see “SQL Functions” in the *SQL Anywhere Server – SQL Reference*.

**Note** Sort-key values created using a version of Sybase IQ earlier than 15.0 do not contain the same values created using version 15.0 and later. This may be a problem for your applications if your pre-15.0 database has sort-key values stored within it, especially if sort-key value comparison is required by your application. Regenerate any sort-key values in your database that were generated using a version of Sybase IQ earlier than 15.0.

---

**SOUNDEX function [String]**

**Function**

Returns a number representing the sound of a string.

**Syntax**

```
SOUNDEX( string-expression )
```

**Parameters**

`string-expression` The string.

**Example**

The following statement returns two numbers, representing the sound of each name. The SOUNDEX value for each argument is 3827.

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' ) FROM iq_dummy
```

SOUNDEX( 'Smith' ) is equal to SOUNDEX( 'Smythe' ).

**Usage**

The SOUNDEX function value for a string is based on the first letter and the next three consonants other than H, Y, and W. Doubled letters are counted as one letter. For example:

---

Reference: Building Blocks, Tables, and Procedures
Alphabetical list of functions

SOUNDEX('apples') FROM iq_dummy

is based on the letters A, P, L, and S.

Multibyte characters are ignored by the SOUNDEX function.

Although it is not perfect, SOUNDEX normally returns the same number for words that sound similar and that start with the same letter.

Standards and compatibility

- SQL92  Vendor extension
- Sybase  Compatible with Adaptive Server Enterprise, except that Adaptive Server Enterprise returns a CHAR(4) result and Sybase IQ returns an integer

SPACE function [String]

Function  Returns a specified number of spaces.
Syntax    SPACE(integer-expression)
Parameters
integer-expression  The number of spaces to return.
Example   The following statement returns a string containing 10 spaces:

SELECT SPACE( 10 ) FROM iq_dummy

Standards and compatibility

- SQL92  Vendor extension
- Sybase  Compatible with Adaptive Server Enterprise

SQRT function [Numeric]

Function  Returns the square root of a number.
Syntax    SQRT(numeric-expression)
Parameters
numeric-expression  The number for which the square root is to be calculated.
Example   The following statement returns the value 3:

SELECT SQRT( 9 ) FROM iq_dummy

Standards and compatibility

- SQL92  Vendor extension
- Sybase  Compatible with Adaptive Server Enterprise
CHAPTER 4  SQL Functions

SQUARE function [Numeric]

Function  Returns the square of the specified expression as a float.
Syntax  \[ SQUARE \ ( \text{numeric-expression} ) \]
Parameters  expression  Is a column, variable, or expression with a data type that is either exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types. For other data types, the SQUARE function generates an error. The return value is of \text{DOUBLE} data type.
Usage  SQUARE function takes one argument. For example, \( \text{SQUARE}(12.01) \) returns 144.240100.
Standards and compatibility  • SQL92  Vendor extension
  • Sybase  Compatible with Adaptive Server Enterprise

STDDEV function [Aggregate]

Function  Returns the standard deviation of a set of numbers.
Syntax  \[ \text{STDDEV} \ (\ [\text{ALL} ] \ \text{expression}) \]
Parameters  expression  Any numeric data type (FLOAT, REAL, or DOUBLE precision) expression.
Examples  Given this data:

\[
\begin{array}{c}
\text{Salary} \\
51432.000 \\
57090.000 \\
42300.000 \\
43700.00 \\
36500.000 \\
138948.000 \\
31200.000 \\
58930.00 \\
75400.00
\end{array}
\]

The following statement returns the value 32617.8446712838471:

\[
\begin{array}{c}
\text{SELECT STDDEV (Salary) FROM Employees WHERE DepartmentID = 300} \\
\end{array}
\]
Given this data:

```sql
SELECT UnitPrice FROM Products WHERE Name = 'Tee Shirt'
```

<table>
<thead>
<tr>
<th>Name</th>
<th>UnitPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tee Shirt</td>
<td>9.00</td>
</tr>
<tr>
<td>Tee Shirt</td>
<td>14.00</td>
</tr>
<tr>
<td>Tee Shirt</td>
<td>14.00</td>
</tr>
</tbody>
</table>

The following statement returns the value 2.88675134594813049:

```sql
SELECT STDDEV ( UnitPrice ) FROM Products
WHERE Name = 'Tee Shirt'
```

Usage

The formula used to calculate \( \text{STDDEV} \) is:

\[
\text{stddev} = \sqrt{\text{variance}}
\]

STDDEV returns a result of data type DOUBLE precision floating-point. If applied to the empty set, the result is NULL, which returns NULL for a one-element input set.

STDDEV does not support the keyword DISTINCT. A syntax error is returned if you use `DISTINCT` with `STDDEV`.

Standards and compatibility

- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

- “STDDEV_SAMP function [Aggregate]” on page 259
- “VARIANCE function [Aggregate]” on page 333
- Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

---

**STDDEV_POP function [Aggregate]**

Function Computes the standard deviation of a population consisting of a numeric-expression, as a DOUBLE.

Syntax

\[
\text{STDDEV\_POP} ( [ \text{ALL} ] \text{expression} )
\]

Parameters

- **expression** The expression (commonly a column name) whose population-based standard deviation is calculated over a set of rows.

Examples

The following statement lists the average and variance in the number of items per order in different time periods:
SELECT year( ship_date ) AS Year, quarter( ship_date ) AS Quarter, AVG( quantity ) AS Average, STDDEV_POP( quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
ORDER BY Year, Quarter;

Usage
Computes the population standard deviation of the provided value expression evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the population variance.

\[ \sqrt{\frac{\sum (x_i - \bar{x})^2}{n}} \]

Standards and compatibility
- **SQL99**  SQL/foundation feature outside of core SQL.
- **SQL92**  Vendor extension
- **Sybase**  Not supported by Adaptive Server Enterprise

See also
“Analytical functions” on page 104

**STDDEV_SAMP function [Aggregate]**

Function
Computes the standard deviation of a sample consisting of a numeric-expression, as a DOUBLE.

Note  STDDEV_SAMP is an alias for STDDEV.

Syntax

```
STDDEV_SAMP ( [ ALL ] expression )
```

Parameters
- **expression**  The expression (commonly a column name) whose sample-based standard deviation is calculated over a set of rows.

Examples
The following statement lists the average and variance in the number of items per order in different time periods:
Alphabetical list of functions

```sql
SELECT year( ship_date ) AS Year, quarter( ship_date )
    AS Quarter, AVG( quantity ) AS Average,
    STDDEV_SAMP( quantity ) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Average</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1</td>
<td>25.775148</td>
<td>14.3218</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>27.050847</td>
<td>15.0696</td>
</tr>
</tbody>
</table>

Usage

Computes the sample standard deviation of the provided value expression evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the sample variance.

NULL returns NULL for a one-element input set.

Standard deviations are computed according to the following formula, which assumes a normal distribution:

\[
\sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}
\]

Standards and compatibility

- **SQL99** SQL/foundation feature outside of core SQL.
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

- “Analytical functions” on page 104
- “STDDEV function [Aggregate]” on page 257
- Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

---

**STR function [String]**

Function

Returns the string equivalent of a number.

Syntax

```
STR( numeric-expression [, length [, decimal] ] )
```

Parameters

- `numeric-expression` Any approximate numeric (FLOAT, REAL, or DOUBLE precision) expression.
**length**  The number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, the sign, if any, and blanks). The default is 10 and the maximum length is 255.

**decimal**  The number of digits to the right of the decimal point to be returned. The default is 0.

**Examples**

The following statement returns a string of six spaces followed by 1234, for a total of ten characters:

```sql
SELECT STR( 1234.56 ) FROM iq_dummy
```

The following statement returns the result 1234.5:

```sql
SELECT STR( 1234.56, 6, 1 ) FROM iq_dummy
```

**Usage**

If the integer portion of the number cannot fit in the length specified, then the result is NULL. For example, the following statement returns NULL:

```sql
SELECT STR( 1234.56, 3 ) FROM iq_dummy
```

**Standards and compatibility**

- **SQL92**  Vendor extension
- **Sybase**  Compatible with Adaptive Server Enterprise

---

**STR_REPLACE function [String]**

**Function**

Takes three arguments as input of type BINARY or STRING and replaces any instances of the second string expression (string_expr2) that occur within the first string expression (string_expr1) with a third expression (string_expr3). STR_REPLACE is an alias of REPLACE function

**Syntax**

```sql
REPLACE( string_expr1, string_expr2, string_expr3 )
```

**Parameters**

- **string_expr1**  is the source string, or the string expression to be searched, expressed as CHAR, VARCHAR, UNICHAR, UNIVARCHAR, VARBINARY, or BINARY data type.

- **string_expr2**  is the pattern string, or the string expression to find within the first expression (string_expr1) and is expressed as CHAR, VARCHAR, UNICHAR, UNIVARCHAR, VARBINARY, or BINARY data type.

- **string_expr3**  is the replacement string expression, expressed as CHAR, VARCHAR, UNICHAR, UNIVARCHAR, VARBINARY, or BINARY data type.

**Example 1**

Replaces the string def within the string cdefghi with yyy.

```sql
select replace("cdefghi", "def", "yyy")
```

---

Reference: Building Blocks, Tables, and Procedures 261
Example 2
Replaces all spaces with “toyota”

```sql
select str_replace("chevy, ford, mercedes", ",", "toyota")
----------
chevy, toyotaford, toyotamercedes
```

(1 row(s) affected)

Example 3
Accepts NULL in the third parameter and treats it as an attempt to replace `string_expr2` with NULL, effectively turning `STR_REPLACE` into a “string cut” operation. Returns “abcghijklm”:

```sql
select str_replace("abcdefghijklm", "def", NULL)
----------
abcghijklm
```

(1 row affected)

Usage
- Takes any data type as input and returns STRING or BINARY.
  - For example, an empty string passed as an argument (“”) is replaced with one space (“ ”) before further evaluation occurs. This is true for both BINARY and STRING types.
  - All arguments can have a combination of BINARY and STRING data types.
  - The result length may vary, depending upon what is known about the argument values when the expression is compiled. If all arguments are columns or host variables assigned to constants, Sybase IQ calculates the result length as:
    ```sql
    result_length = ((s/p)*(r-p)+s)
    \text{WHERE}
    s = \text{length of source string}
    p = \text{length of pattern string}
    r = \text{length of replacement string}
    \text{IF}\ (r-p) \leq 0, \text{result length} = s
    ```
- If Sybase IQ cannot calculate the result length because the argument values are unknown when the expression is compiled, the result length used is 255.
- `RESULT_LEN` never exceeds 32767.

Permissions
Any user can execute `STR_REPLACE`.

Standards and compatibility
ANSI SQL – Compliance level: Transact-SQL extension
See also  

**Data types**  
CHAR, VARCHAR, UNICHAR, UNIVARCHAR, VARBINARY, or BINARY. See Chapter 3, “SQL Data Types.”

**Function**  
“LENGTH function [String]” on page 197.
For general information about string functions, see “String functions” on page 115.

---

### STRING function [String]

**Function**  
Concatenates one or more strings into one large string.

**Syntax**  
```sql
STRING ( string-expression [, ... ] )
```

**Parameters**  
- `string-expression`: A string.
  
  If only one argument is supplied, it is converted into a single expression. If more than one argument is supplied, they are concatenated into a single string.

  A NULL is treated as an empty string ("").

**Example**  
The following statement returns the value testing123:
```sql
SELECT STRING( 'testing', NULL, 123 )
FROM iq_dummy
```

**Usage**  
Numeric or date parameters are converted to strings before concatenation. You can also use the STRING function to convert any single expression to a string by supplying that expression as the only parameter.

If all parameters are NULL, STRING returns NULL.

**Standards and compatibility**
- SQL92 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise

---

### STRTOUUID function [String]

**Function**  
Converts a string value to a unique identifier (UUID or GUID) value.

**Syntax**  
```sql
STRTOUUID ( string-expression )
```

**Parameters**  
- `string-expression`: A string in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`

**Example**  
The following creates a table with a uniqueidentifier primary key:
```sql
CREATE TABLE T (  
    pk uniqueidentifier primary key,  
    c1 int);
```
INSERT INTO T (pk, c1)
VALUES (STRTOUUID
('12345678-1234-5678-9012-123456789012'), 1);

Usage
Converts a string in the format xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx where x is a hexadecimal digit, to a unique identifier value. If the string is not a valid UUID string, NULL is returned.

You can use STRTOUUID to insert UUID values into a Sybase IQ database.

Standards and compatibility
• SQL92 Vendor extension
• SQL99 Vendor extension
• Sybase Not supported by Adaptive Server Enterprise

See also
“NEWID function [Miscellaneous]” on page 208
“UUIDTOSTR function [String]” on page 331
UNIQUEIDENTIFIER in “Binary data types” on page 77

STUFF function [String]
Function
Deletes a number of characters from one string and replaces them with another string.

Syntax
STUFF ( string-expression1, start, length, string-expression2 )

Parameters
string-expression1 The string to be modified by the STUFF function.
start The character position at which to begin deleting characters. The first character in the string is position 1.
length The number of characters to delete.
string-expression2 The string to be inserted.

Example
The following statement returns the value “chocolate pie”:
SELECT STUFF('chocolate cake', 11, 4, 'pie')
FROM iq_dummy

Usage
To delete a portion of a string using STUFF, use a replacement string of NULL.
To insert a string using STUFF, use a length of zero.

Standards and compatibility
• SQL92 Vendor extension
• Sybase Compatible with Adaptive Server Enterprise

See also
“INSERTSTR function [String]” on page 189
SUBSTRING function [String]

Function Returns a substring of a string.

Syntax `{ SUBSTRING | SUBSTR } ( string-expression, start [, length ] )`

Parameters
- `string-expression` The string from which a substring is to be returned.
- `start` The start position of the substring to return, in characters. A negative starting position specifies a number of characters from the end of the string instead of the beginning. The first character in the string is at position 1.
- `length` The length of the substring to return, in characters. A positive length specifies that the substring ends length characters to the right of the starting position, while a negative length specifies that the substring ends length characters to the left of the starting position.

Examples
- The following statement returns “back”:
  ```sql
  SELECT SUBSTRING ( 'back yard', 1 , 4 )
  FROM iq_dummy
  ```
- The following statement returns yard:
  ```sql
  SELECT SUBSTR ( 'back yard', -1 , -4 )
  FROM iq_dummy
  ```
- The following statement returns 0x2233:
  ```sql
  SELECT SUBSTR ( 0x112233445566, 2, 2 )
  FROM iq_dummy
  ```

Usage
- If `length` is specified, the substring is restricted to that length. If no length is specified, the remainder of the string is returned, starting at the `start` position.
- Both `start` and `length` can be negative. Using appropriate combinations of negative and positive numbers, you can get a substring from either the beginning or end of the string.

Standards and compatibility
- SQL92 Vendor extension
- Sybase SUBSTR is not supported by Adaptive Server Enterprise. Use SUBSTRING instead.

SUM function [Aggregate]

Function Returns the total of the specified expression for each group of rows.

Syntax `SUM ( expression | DISTINCT column-name )`

Parameters
- `expression` The object to be summed. This is commonly a column name.
**DISTINCT column-name**  Computes the sum of the unique values in `column-name` for each group of rows. This is of limited usefulness, but is included for completeness.

**Example**

The following statement returns the value 3749146.740:

```
SELECT SUM( salary )
FROM Employees
```

**Usage**

Rows where the specified expression is NULL are not included.

Returns NULL for a group containing no rows.

**Standards and compatibility**

- **SQL92**  SQL92 compatible.
- **Sybase**  Compatible with Adaptive Server Enterprise.

**See also**

- “AVG function [Aggregate]” on page 131
- “COUNT function [Aggregate]” on page 146
- Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

---

**SUSER_ID function [System]**

**Function**

Returns an integer user identification number.

**Syntax**

```
SUSER_ID ([ user-name ])
```

**Parameters**

- **user-name**  The user name.

**Examples**

The following statement returns the user identification number 1:

```
SELECT SUSER_ID ( 'DBA' ) FROM iq_dummy
```

The following statement returns the user identification number 0:

```
SELECT SUSER_ID ( 'SYS' ) FROM iq_dummy
```

**Standards and compatibility**

- **SQL92**  Vendor extension
- **Sybase**  Adaptive Server Enterprise function implemented for Sybase IQ

**See also**

- “SUSER_NAME function [System]” on page 266
- “USER_ID function [System]” on page 330

---

**SUSER_NAME function [System]**

**Function**

Returns the user name.
CHAPTER 4    SQL Functions

SUSER_NAME ([user-id])

Syntax

Parameters

user-id  The user identification number.

Examples

The following statement returns the value DBA:

```
SELECT SUSER_NAME (1) FROM iq_dummy
```

The following statement returns the value SYS:

```
SELECT SUSER_NAME (0) FROM iq_dummy
```

Standards and compatibility

• SQL92  Vendor extension
• Sybase  Adaptive Server Enterprise function implemented for Sybase IQ. In Adapter Server Enterprise, SUSER_NAME returns the server user name.

See also

“SUSER_ID function [System]” on page 266
“USER_NAME function [System]” on page 330

TAN function [Numeric]

Function

Returns the tangent of a number.

Syntax

```
TAN ( numeric-expression )
```

Parameters

numeric-expression  An angle, in radians.

Example

The following statement returns the value 0.572561:

```
SELECT TAN(0.52) FROM iq_dummy
```

Standards and compatibility

• SQL92  Vendor extension
• Sybase  Compatible with Adaptive Server Enterprise

See also

“COS function [Numeric]” on page 143
“SIN function [Numeric]” on page 250

TODAY function [Date and time]

Function

Returns the current date. This is the historical syntax for CURRENT DATE.

Syntax

```
TODAY ( )
```

Example

The following statement returns the current day according to the system clock.

```
SELECT TODAY( ) FROM iq_dummy
```
TRIM function [String]

Function

Removes leading and trailing blanks from a string.

Syntax

```
TRIM ( string-expression )
```

Parameters

- `string-expression` The string to be trimmed.

Note

The result data type of a TRIM function is a LONG VARCHAR. If you use TRIM in a SELECT INTO statement, you must have a Large Objects Management option license, or use CAST and set TRIM to the correct data type and size.

Example

The following statement returns the value “chocolate” with no leading or trailing blanks.

```
SELECT TRIM( '   chocolate   ' ) FROM iq_dummy
```
### CHAPTER 4  SQL Functions

#### Examples

The following statement returns the value 600:

```
SELECT TRUNCNUM( 655, -2 ) FROM iq_dummy
```

The following statement: returns the value 655.340:

```
SELECT TRUNCNUM( 655.348, 2 ) FROM iq_dummy
```

#### Usage

This function is the same as TRUNCATE, but does not cause keyword conflicts.

You can use combinations of ROUND, FLOOR, and CEILING to provide similar functionality.

#### Standards and compatibility

- **SQL92**  Vendor extension
- **Sybase**  Not supported in Adaptive Server Enterprise

#### See also

“ROUND function [Numeric]” on page 245

---

**TS_ARMA_AR function [Time Series]**

*Note*  This function is available only with RAP – The Trading Edition Enterprise.

**Function**

Calculates the least-square estimates of parameters for an autoregressive moving average (ARMA) model, and returns the requested autoregressive estimate.

**Syntax**

```
TS_ARMA_AR (timeseries_expression, ar_count, ar_elem, method)
OVER (window-spec)
```

**Parameters**

- **timeseries_expression**  A numeric expression, generally a column name, containing an element in a time series.
- **ar_count**  An integer containing the number of autoregressive values to compute.
- **ar_elem**  An integer identifying the element in the computed AR array that should be returned. `ar_elem` must be greater than 0 and less than or equal to `ar_count`.
- **method**  (Optional) An integer that identifies the type of procedure used to compute estimates. 0 (the default value) = method of least squares; 1 = method of moments.
- **window-spec**  `TS_ARMA_AR` is an OLAP function requiring an `OVER ()` clause.
Usage

TS_ARMA_AR time series function returns a double-precision floating-point value containing the autoregressive estimate. TS_ARMA_AR calls the function imsls_d_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_ARMA_AR map to the IMSL library function imsls_d_arma as follows:

\[
\text{params} = \text{imsls_d_arma}(n\_\text{objs}, z, p, q, \text{methodID}, 0);
\]

- **n_objs** Contains the number of rows in the current window frame.
- **z[]** Contains the value of timeseries_expression for the current window frame.
- **p** Maps to the user-defined aggregate function argument ar_count.
- **q** = 1.
- **methodID** Maps to the method argument of TS_ARMA_AR. Can be set to either IMSLS_METHOD_OF_MOMENTS or IMSLS_LEAST_SQUARES.

For detailed information on how imsls_d_arma performs time series calculations, see IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library.

Example

This example shows an input data table, a SQL statement containing the TS_ARMA_AR function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data.

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>rownum</td>
<td>data</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>
Alphabetical list of functions

The following SQL statement returns the first element of an autoregressive estimate consisting of one value from the data column using the method of least squares:

```sql
SELECT TS_ARMA_AR(data,1,1,0) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.898793</td>
</tr>
<tr>
<td>0.898793</td>
</tr>
<tr>
<td>0.898793</td>
</tr>
<tr>
<td>0.898793</td>
</tr>
<tr>
<td>0.898793</td>
</tr>
<tr>
<td>0.898793</td>
</tr>
<tr>
<td>0.898793</td>
</tr>
<tr>
<td>0.898793</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0.898793</td>
</tr>
</tbody>
</table>

Standards and compatibility

- **SQL2008**  Sybase extension
- **Sybase**  Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also


C Stat Library

---

**TS_ARMA_CONST function [Time Series]**

**Note**  This function is available only with RAP – The Trading Edition Enterprise.

**Function**  Calculates the least-square estimates of parameters for an autoregressive moving average (ARMA) model, and returns an estimated constant.
Syntax

```
TS_ARMA_CONST (timeseries_expression, method)
OVER (window-spec)
```

Parameters

- **timeseries_expression**: A numeric expression, generally a column name, containing an element in a time series.
- **method**: An integer that identifies the type of procedure used to compute estimates. 0 (the default value) = method of least squares; 1 = method of moments.
- **window-spec**: TS_ARMA_CONST is an OLAP function requiring an OVER () clause.

Usage

This time series function returns a double-precision floating-point value containing the constant estimate produced by the function. TS_ARMA_CONST calls the function imsls_d_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_ARMA_CONST map to the IMSL library function imsls_d_arma as follows:

```
params = imsls_d_arma(n_objs, z, p, q, IMSLS_CONSTANT, method_id, 0);
```

- **n_objs**: Contains the number of rows in the current window frame.
- **z[]**: Contains the value of timeseries_expression for the current window frame.
- **p =1.**
- **q =1.**

**MethodID**

Maps to the method argument of TS_ARMA_CONST.

For detailed information on how the function imsls_d_arma performs time series calculations, see *IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library*.

Example 1

This example shows an input data table, a SQL statement containing the TS_ARMA_CONST function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
</tbody>
</table>

Reference: Building Blocks, Tables, and Procedures
### Alphabetical list of functions

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505111</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
</tbody>
</table>
The following SQL statement returns an estimated constant from the data column using the method of least squares:

```
SELECT TS_ARMA_CONST(data,0) OVER (ORDER BY ROWNUM rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

**Table 4-30: Values returned from TS_ARMA_CONST Example 1**

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.082077</td>
</tr>
<tr>
<td>0.082077</td>
</tr>
<tr>
<td>0.082077</td>
</tr>
<tr>
<td>0.082077</td>
</tr>
<tr>
<td>0.082077</td>
</tr>
<tr>
<td>0.082077</td>
</tr>
<tr>
<td>0.082077</td>
</tr>
<tr>
<td>0.082077</td>
</tr>
<tr>
<td>0.082077</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0.082077</td>
</tr>
</tbody>
</table>

**Example 2**

This example provides a sample query that returns estimates for the AR, MA, and constant parameters. The first element for AR and MA in the array contains one element. See Table 4-29 on page 273 for the DATASET table.

```
SELECT TS_ARMA_AR(DATA,1,1,0) OVER (ORDER BY rownum rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS ar_param, ts_arma_ma(data,1,1,0) OVER ... FROM DATASET
```

Reference: Building Blocks, Tables, and Procedures
Sybase IQ returns 50 rows of data, each containing the same three values:

<table>
<thead>
<tr>
<th>ar_param</th>
<th>ma_param</th>
<th>const_param</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.898793</td>
<td>0.105075</td>
<td>0.082077</td>
</tr>
<tr>
<td>0.898793</td>
<td>0.105075</td>
<td>0.082077</td>
</tr>
<tr>
<td>0.898793</td>
<td>0.105075</td>
<td>0.082077</td>
</tr>
<tr>
<td>0.898793</td>
<td>0.105075</td>
<td>0.082077</td>
</tr>
<tr>
<td>0.898793</td>
<td>0.105075</td>
<td>0.082077</td>
</tr>
<tr>
<td>0.898793</td>
<td>0.105075</td>
<td>0.082077</td>
</tr>
<tr>
<td>0.898793</td>
<td>0.105075</td>
<td>0.082077</td>
</tr>
<tr>
<td>0.898793</td>
<td>0.105075</td>
<td>0.082077</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0.898793</td>
<td>0.105075</td>
<td>0.082077</td>
</tr>
</tbody>
</table>

Standards and compatibility

- **SQL2008**  Sybase extension
- **Sybase**  Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

---

**TS_ARMA_MA function [Time Series]**

**Note**  This function is available only with RAP – The Trading Edition Enterprise.

**Function**  Calculates the least-square estimates of parameters for an autoregressive moving average (ARMA) model, and returns the requested moving average estimate.

**Syntax**

```sql
TS_ARMA_MA (timeseries_expression, ma_count, ma_elem, method) OVER (window-spec)
```
Parameters

- **timeseries_expression**: A numeric expression, generally a column name, containing an element in a time series.
- **ma_count**: An integer containing the number of autoregressive values to compute.
- **ma_elem**: An integer identifying the element to return from the computed moving average array. The integer must be greater than 0 and less than or equal to **ma_count**.
- **method**: (Optional) An integer identifying the procedure to use to calculate estimates. 0 (the default value) = method of least squares and 1 = method of moments.
- **window-spec**: TS_ARMA_MA is an OLAP function requiring an OVER () clause.

Usage

This time series function returns a double-precision floating-point value representing the moving average estimate. TS_ARMA_MA calls the function imsls_d_arma in the IMSL libraries.

IMSL mapping

The arguments of TS_ARMA_MA map to the IMSL library function imsls_d_arma as follows:

```java
params = imsls_d_arma(n_objs, z, p, q, method_id, 0);
```

- **n_objs**: Contains the number of rows in the current window frame.
- **z[]**: Contains the value of **timeseries_expression** for the current window frame.
- **p** = 1.
- **q**: Maps to the user-defined aggregate function argument **ma_count**.
- **method_id**: Maps to the **method** argument of TS_ARMA_MA.

For detailed information on how the function imsls_d_arma performs time series calculations, see *IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the TS_ARMA_MA function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.311523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
</tbody>
</table>

*Table 4-32: Input data table DATASET*
### Alphabetical list of functions

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
</tbody>
</table>
The following SQL statement returns the first element of an array containing one element from the data column using the method of least squares:

```sql
SELECT TS_ARMA_MA(data,1,1,0) OVER (ORDER BY rownum
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
</tbody>
</table>

The following SQL statement returns the first element of an array containing one element from the data column using the method of least squares:

```sql
SELECT TS_ARMA_MA(data,1,1,0) OVER (ORDER BY rownum
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

Table 4-33: Values returned from TS_ARMA_MA

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
<tr>
<td>0.105075</td>
</tr>
</tbody>
</table>

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

Reference: Building Blocks, Tables, and Procedures
TS_AUTOCORRELATION function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function Calculates the sample autocorrelation function of a stationary time series.

Syntax

```
TS_AUTOCORRELATION (timeseries_expression, lagmax, lag_elem)
```

OVER (window-spec)

Parameters

- **timeseries_expression** A numeric expression, generally a column name, containing an element in a time series.
- **lagmax** An integer specifying the maximum lag of autocovariances, autocorrelations, and standard errors of autocorrelations. The integer must be greater than or equal to 1, and less than the number of elements in the time series.
- **lag_elem** An integer specifying which element in the autocorrelation array is to be returned. The integer must be greater than zero, and less than or equal to lagmax.
- **window-spec** TS_AUTOCORRELATION is an OLAP function requiring an OVER () clause.

Usage

This time series function returns a double-precision floating-point value representing the autocorrelation value. TS_AUTOCORRELATION calls the function imsls_d_autocorrelation in the IMSL libraries.

IMSL mapping

The arguments of TS_AUTOCORRELATION map to the IMSL library function imsls_d_autocorrelation() as follows:

```
params = imsls_d_autocorrelation(n_objs, x[], lagmax, 0);
```

- **n_objs** Contains the number of rows in the current window frame.
- **x[]** Contains the value of timeseries_expression for the current window frame.
- **lagmax** Maps to the user-defined aggregate function argument lag_max.
For detailed information on how the function `imsls_d_autocorrelation` performs time series calculations, see *IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library*.

This example shows an input data table, a SQL statement containing the `TS_AUTOCORRELATION` function, and the data values returned by the function. This example uses the following table (called `DATASET`) as its input data. The `DATASET` table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77355</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
</tbody>
</table>
The following SQL statement returns the second element from an array containing autocorrelations of the time series data from the `data` column:

```sql
SELECT TS_AUTOCORRELATION(data,2,2) OVER (ORDER BY ROWNUM rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

**Table 4-35: Values returned from TS_AUTOCORRELATION**

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>

The **Table 4-35: Values returned from TS_AUTOCORRELATION**
Standards and compatibility
- SQL2008: Sybase extension
- Sybase: Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

TS_AUTO_UNI_AR function [Time Series]

Note: This function is available only with RAP – The Trading Edition Enterprise.

Function
Performs automatic selection and fitting of a univariate autoregressive time series model.

Syntax
```
TS_AUTO_UNI_AR (timeseries_expression, ar_count, ar_elem, method)
OVER (window-spec)
```

Parameters
- **timeseries_expression**: A numeric expression, generally a column name, containing an element in a time series.
- **ar_count**: An integer containing the number of autoregressive values to compute.
- **ar_elem**: An integer identifying which computed autoregressive value to return. The integer must be greater than zero, and less than or equal to ar_count.
- **method**: (Optional) An integer identifying which method to use when computing AR coefficients, where:
  0 = method of moments
  1 = method of least squares (the default value)
  2 = maximum likelihood.
window-spec  TS_AUTO_UNI_AR is an OLAP function requiring an OVER ()
clause.

Usage
This time series function returns a double-precision floating-point number
containing the autoregressive estimate. TS_AUTO_UNI_AR calls the function
imsls_d_auto_uni_ar in the IMSL libraries.

IMSL mapping
The arguments of TS_AUTO_UNI_AR map to the IMSL library function
imsls_d_auto_uni_ar as follows:

\[ \text{params} = \text{imsls_d_auto_uni_ar} (n\_objs, z[], \text{maxlag}, p, \]
\[ \text{method}, 0); \]

n_objs  Contains the number of rows in the current window frame.

z[]  Contains the value of timeseries_expression for the current window
frame.

maxlag  Maps to the user-defined aggregate function argument ar_count.

p  The output parameter, representing the number of autoregressive
parameters in the model with minimum AIC.

method  Maps to the user-defined aggregate function argument method. If
ar_elem is greater than p, and if your IMSL library time series function error-
handling value is set to 0, IQ returns a null. If your IMSL library time series
function error-handling value is set to a value other than 0, IQ displays an error
message indicating that ar_elem is greater than p. See “IMSL library time
series function error-handling” on page 125.

For detailed information on how the function imsls_d_auto_uni_ar performs
time series calculations, see IMSL Numerical Library User’s Guide: Volume 2
of C Stat Library.

Example
This example shows an input data table, a SQL statement containing the
TS_AUTO_UNI_AR function, and the data values returned by the function. This
example uses the following table (called DATASET) as its input data. The
DATASET table contains 50 rows of time series data:

table 4-36: Input data table DATASET

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>rownum</td>
<td>data</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
</tbody>
</table>
The following SQL statement returns the first element from an array containing two elements from the data column:

```
SELECT TS_AUTO_UNI_AR(data,2,1,0) OVER (ORDER BY ROWNUM
rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

**Table 4-37: Values returned from TS_AUTO_UNI_AR**

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.883453</td>
</tr>
<tr>
<td>0.883453</td>
</tr>
<tr>
<td>0.883453</td>
</tr>
<tr>
<td>0.883453</td>
</tr>
<tr>
<td>0.883453</td>
</tr>
<tr>
<td>0.883453</td>
</tr>
<tr>
<td>0.883453</td>
</tr>
<tr>
<td>0.883453</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0.883453</td>
</tr>
</tbody>
</table>

Standards and compatibility

- **SQL2008**  Sybase extension
- **Sybase**  Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

TS_BOX_COX_XFORM function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function Performs a forward or inverse Box-Cox power transformation.

Syntax

```
TS_BOX_COX_XFORM (timeseries_expression, power [, shift [, inverse] ] ) OVER (window-spec)
```

Parameters

- `timeseries_expression` A numeric expression, generally a column name, containing an element in a time series.
- `power` A double-precision floating-point value representing an exponent parameter in the Box-Cox power transformation.
- `shift` (Optional) A double-precision floating-point value representing a shift parameter. The value must satisfy the relation:
  \[ \text{min}(\text{timeseries}) + \text{shift} > 0 \]
  `shift` defaults to 0.0.
- `inverse` (Optional) A tinyint value; if set to 1, IQ performs an inverse transformation. If 0 or null, IQ performs a forward transformation. The default value is 0.

window-spec TS_BOX_COX_XFORM is an OLAP function requiring an OVER () clause with an unbounded window. TS_BOX_COX_XFORM does not support value-based windows; for example, you cannot use a range specifier in the OVER () clause.

Usage TS_BOX_COX_XFORM returns the corresponding calculated transformed value for each element in the time series; it calls the function `imsls_d_box_cox_transform` in the IMSL libraries.

IMSL mapping The arguments of TS_BOX_COX_XFORM map to the IMSL library function `imsls_d_box_cox_transform` as follows:

```
params = imsls_d_box_cox_transform(n_objs, z[ ], power, IMSLS_SHIFT, shift [, IMSLS_INVERSE], 0);
```

- `n_objs` Contains the number of rows in the current window frame.
- `z[]` Contains the value of `timeseries_expression` for the current window frame.
- `power` Maps to the user-defined aggregate function argument `power`.
- `shift` Maps to the user-defined aggregate function argument `shift`.
**Example**

**IMSLS_INVERSE**  If the user-defined aggregate function argument *inverse* is 1, IQ calls the Box-Cox transform with IMSLS_INVERSE, otherwise this argument is left out of the function call.

For detailed information on how the function imsls_d_box_cox_transform performs time series calculations, see *IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library*.

This example shows an input data table, a SQL statement containing the TS_BOX_COX_XFORM function, and the data values returned by the function. This example uses the following table (called BOX_COX_XFORM_DATASET) as its input data. The BOX_COX_XFORM_DATASET table contains 13 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>78.5</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>52</td>
</tr>
<tr>
<td>10</td>
<td>74.3</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>56</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
</tr>
</tbody>
</table>

The following SQL statement returns the Box-Cox power transformation from the data column:

```
SELECT TS_BOX_COX_XFORM(data,1.0,1.0,0) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM BOX_COX_XFORM_DATASET
```

Sybase IQ returns the following 13 rows:

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

Table 4-38: Input data table BOX_COX_XFORM_DATASET

Table 4-39: Values returned from TS_BOX_COX_XFORM
### Standards and compatibility
- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

### See also

---

### TS_DIFFERENCE function [Time Series]

**Note** This function is available only with RAP – The Trading Edition Enterprise.

**Function**
Differences a seasonal or nonseasonal time series.

**Syntax**
```sql
TS_DIFFERENCE (timeseries_expression, period1 [, period2 [, ...period 10] ] ) OVER (window-spec)
```

**Parameters**
- `timeseries_expression` A numeric expression, generally a column name, containing an element in a time series to be differenced.
- `period1 ... period10` Each period is an integer expression containing the period in which the time series is to be differenced. You must specify at least one period, and you can specify up to 10 periods.
- `window-spec` `TS_DIFFERENCE` is an OLAP function requiring an `OVER ()` clause with an unbounded window. This function does not support value-based windows; for example, you cannot use a range specifier in the `OVER ()` clause.

---

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
</tr>
<tr>
<td>79.5</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>53</td>
</tr>
<tr>
<td>75.3</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>57</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>
Usage

For each element in the time series, TS_DIFFERENCE returns the corresponding calculated differenced value for the time series; it calls the function imsls_d_difference in the IMSL libraries.

IMSL mapping

The arguments of TS_DIFFERENCE map to the IMSL library function imsls_d_difference as follows:

\[
\text{params} = \text{imsls_d_difference}(\text{n_objs}, z[,], \text{n_differences}, \text{periods }[,], 0);
\]

- **n_objs**  Contains the number of rows in the current window frame.
- **z[]**     Contains the value of timeseries_expression for the current window frame.
- **n_differences**  Maps to the period arguments defined in TS_DIFFERENCE.
- **period**  An array of the period arguments defined in TS_DIFFERENCE.

For detailed information on how the function imsls_d_difference performs time series calculations, see IMSL C Numerical Library User’s Guide Volume 2 of 2: C Stat Library.

Example

This example shows an input data table, a SQL statement containing the TS_DIFFERENCE function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>rownum</td>
<td>data</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>

The following SQL statement differences data from the data column:
Alphabetical list of functions

SELECT TS_DIFFERENCE(data,1) OVER (ORDER BY ROWNUM rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET

Sybase IQ returns 50 rows:
Table 4-41: Values returned from TS_DIFFERENCE

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>0.170336</td>
</tr>
<tr>
<td>0.191027</td>
</tr>
<tr>
<td>1.29692</td>
</tr>
<tr>
<td>0.801743</td>
</tr>
<tr>
<td>-0.038988</td>
</tr>
<tr>
<td>-0.09424</td>
</tr>
<tr>
<td>1.61886</td>
</tr>
<tr>
<td>-1.12477</td>
</tr>
<tr>
<td>1.02925</td>
</tr>
<tr>
<td>-1.20614</td>
</tr>
<tr>
<td>-0.814478</td>
</tr>
<tr>
<td>-0.157049</td>
</tr>
<tr>
<td>-1.18213</td>
</tr>
<tr>
<td>0.027546</td>
</tr>
<tr>
<td>1.45734</td>
</tr>
<tr>
<td>-0.990302</td>
</tr>
<tr>
<td>-0.833325</td>
</tr>
<tr>
<td>-0.637229</td>
</tr>
<tr>
<td>-0.08655</td>
</tr>
<tr>
<td>-0.12594</td>
</tr>
<tr>
<td>-0.122914</td>
</tr>
<tr>
<td>-1.39596</td>
</tr>
<tr>
<td>0.919785</td>
</tr>
<tr>
<td>-0.449474</td>
</tr>
<tr>
<td>0.037273</td>
</tr>
<tr>
<td>-0.954345</td>
</tr>
<tr>
<td>-0.562983</td>
</tr>
<tr>
<td>1.98379</td>
</tr>
<tr>
<td>0.88304</td>
</tr>
<tr>
<td>-0.345265</td>
</tr>
<tr>
<td>0.934656</td>
</tr>
<tr>
<td>0.069088</td>
</tr>
<tr>
<td>-0.249428</td>
</tr>
<tr>
<td>0.795766</td>
</tr>
<tr>
<td>-1.8145</td>
</tr>
</tbody>
</table>
Alphabetical list of functions

TS_ESTIMATE_MISSING function [Time Series]

Note This function is available only with RAP – The Trading Edition Enterprise.

Function Estimates the missing values in a time series and returns them as a new time series, interspersed with the original time series.

Syntax `TS_ESTIMATE_MISSING(timeseries_expression, method)`

Note The first row of results is NULL because the IMSL library returned a not a number (NaN) value for that row.

Standards and compatibility

- SQL2008 Sybase extension
- Sybase Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also Chapter 2, “Using OLAP” in the System Administration Guide: Volume 2


<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.27016</td>
</tr>
<tr>
<td>1.39266</td>
</tr>
<tr>
<td>-0.141794</td>
</tr>
<tr>
<td>0.934752</td>
</tr>
<tr>
<td>0.982506</td>
</tr>
<tr>
<td>0.330772</td>
</tr>
<tr>
<td>-1.34311</td>
</tr>
<tr>
<td>1.23124</td>
</tr>
<tr>
<td>0.209869</td>
</tr>
<tr>
<td>0.791146</td>
</tr>
<tr>
<td>-0.259155</td>
</tr>
<tr>
<td>0.15124</td>
</tr>
<tr>
<td>-0.963484</td>
</tr>
<tr>
<td>0.383186</td>
</tr>
</tbody>
</table>
OVER (window-spec)

Parameters

timeseries_expression A numeric expression, generally a column name, containing an element in a time series to be differenced. If a null-value is provided, it is assumed to reflect a gap in the time series, the value of which will be computed by the function.

method (Optional) An integer specifying the method to use when determining missing values:

- 0 (default) — estimates the missing time series observations in a gap by the median of the last four time series values before and the first four values after the gap.
- 1 — uses a cubic spline interpolation method to estimate missing values. Here, the interpolation is again done over the last four time series values before and the first four values after the gap.
- 2 — assumes that the time series before the gap can be well described by an AR(1) process.
- 3 — uses an AR(p) model to estimate missing values by a one-step-ahead forecast.

window-spec TS_ESTIMATE_MISSING is an OLAP function requiring an OVER () clause with an unbounded window. This function does not support value-based windows; for example, you cannot use a range specifier in the OVER () clause.

Usage

Use TS_ESTIMATE_MISSING to estimate any missing equidistant time points using one of the four estimation methods. TS_ESTIMATE_MISSING calls the function imsls_d_estimate_missing in the IMSL libraries.

You cannot use TS_ESTIMATE_MISSING if more than two consecutive NULL values exist in your set of timepoints. If the first or last two values in the set of timepoints are NULL, the function returns NULL.

IMSL mapping

The arguments of TS_ESTIMATE_MISSING map to the IMSL library function imsls_d_estimate_missing as follows:

```sql
params = imsls_d_estimate_missing(n_objs, tpoints[], z[], method, 0);
```

n_objs Contains the number of rows in the current window frame.

tpoints An array of indexes for specifying missing values in a sequence of timepoints.

z[] The accumulated timeseries_expression, obtained during calls to next_value.
**method**  Maps to the *method* argument defined in `TS_ESTIMATE_MISSING`.

For detailed information on how the function `imsls_d_estimate_missing` performs time series calculations, see *IIMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library*.

This example shows an input data table, a SQL statement containing the `TS_ESTIMATE_MISSING` function, and the data values returned by the function. This example uses the following table (called `EST_MISSING_DATASET`) as its input data. The `EST_MISSING_DATASET` table contains nine rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.8223</td>
</tr>
<tr>
<td>2</td>
<td>-0.5721</td>
</tr>
<tr>
<td>3</td>
<td>2.2771</td>
</tr>
<tr>
<td>4</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>1.2648</td>
</tr>
<tr>
<td>6</td>
<td>1.0278</td>
</tr>
<tr>
<td>7</td>
<td>0.6991</td>
</tr>
<tr>
<td>8</td>
<td>-1.7539</td>
</tr>
<tr>
<td>9</td>
<td>-2.8875</td>
</tr>
</tbody>
</table>

The following SQL statement estimates the value of the data missing from the fourth row:

```sql
SELECT TS_ESTIMATE_MISSING(DATA, 0) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM EST_MISSING_DATASET
```

Sybase IQ returns the following nine rows, replacing the NULL value with 1.0278:

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8223</td>
</tr>
<tr>
<td>-0.5721</td>
</tr>
<tr>
<td>2.2771</td>
</tr>
<tr>
<td>1.0278</td>
</tr>
<tr>
<td>1.2648</td>
</tr>
<tr>
<td>1.0278</td>
</tr>
<tr>
<td>0.6991</td>
</tr>
</tbody>
</table>
Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also


---

**TS_LACK OF FIT function [Time Series]**

**Note** This function is available only with RAP – The Trading Edition Enterprise.

**Function**

Performs the lack-of-fit test for a univariate time series or transfer function, given the appropriate correlation function.

**Syntax**

```
TS_LACK_OF_FIT (timeseries_expression, p_value, q_value, lagmax, [tolerance])
```

```
OVER (window-spec)
```

**Parameters**

- `timeseries_expression` A numeric expression, generally a column name, containing an element in a time series.
- `p_value` An integer containing the number of autoregressive parameters.
- `q_value` An integer containing the number of moving average parameters.
- `lagmax` An integer containing the maximum lag of the correlation function.
- `tolerance` (An optional parameter). A floating-point value level used to determine convergence of the nonlinear least-squares algorithm. The default value is 0.
- `window-spec` TS_LACK_OF_FIT is an OLAP function requiring an OVER () clause.

**Usage**

This function returns a double-precision floating-point value containing the lack-of-fit statistic (q) for the time series. TS_LACK_OF_FIT calls the function `imsls_d_lack_of_fit` in the IMSL libraries.

---

```
res
    -1.7539
    -2.8875
```
Alphabetical list of functions

**IMSL mapping**

The arguments of TS_LACK_OF_FIT map to the IMSL library function `imsls_d_lack_of_fit` as follows:

```c
params = imsls_d_arma(n_objs, z[], p, q,
    IMSLS_LEAST_SQUARES,
    IMSLS_CONVERGENCE_TOLERANCE, tolerance,
    IMSL_RESIDUAL, &residual, 0);
correlations = imsls_d_autocorrelation(n_objs-
p+lagmax, residuals, lagmax, 0);
result = imsls_d_lack_of_fit(n_objs, correlations,
    lagmax, npfree, 0);
```

**n_objs**  Contains the number of rows in the current window frame.

**z[]**  Contains the value of _timeseries_expression_ for the current window frame.

**p**  Maps to the _p_value_ argument defined in TS_LACK_OF_FIT.

**q**  Maps to the _q_value_ argument defined in TS_LACK_OF_FIT.

**lagmax**  Maps to the _lagmax_ argument defined in TS_LACK_OF_FIT.

**npfree**  Derived from _p_ + _q_.

**tolerance**  An optional argument using IMSLS_CONVERGENCE_TOLERANCE. If null, the IMSL library applies a default value and does not use IMSLS_CONVERGENCE_TOLERANCE.

For detailed information on how the IMSL function `imsls_d_lack_of_fit` performs time series calculations, see _IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library._

**Example**

This example shows an input data table, a SQL statement containing the TS_LACK_OF_FIT function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

**Table 4-44: Input data table DATASET**

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>rownum</td>
<td>data</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
</tbody>
</table>
The following SQL statement returns the lack of fit statistic on data from the `data` column:

```
SELECT TS_LACK_OF_FIT(data,1,1,5,0.225) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

### Table 4-45: Values returned from TS_LACK_OF_FIT

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>3.96751</td>
</tr>
</tbody>
</table>

### Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise.

### See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

**TS_LACK OF FIT_P function [Time Series]**

**Function**
Performs the lack-of-fit test for a univariate time series. This function is identical to “TS_LACK OF FIT function [Time Series]” on page 297, except that it returns the p-value of q, rather than returning q.

**Syntax**

```sql
TS_LACK_OF_FIT_P (timeseries_expression, p_value, q_value, lagmax, [tolerance])
OVER (window-spec)
```

**Parameters**
- **timeseries_expression** A numeric expression, generally a column name, containing an element in a time series.
- **p_value** An integer containing the number of autoregressive parameters.
- **q_value** An integer containing the number of moving average parameters.
- **lagmax** An integer containing the maximum log of the correlation function.
- **tolerance** (An optional parameter). A floating-point value level used to determine convergence of the nonlinear least-squares algorithm. The default value is 0.
- **window-spec** TS_LACK_OF_FIT_P is an OLAP function requiring an OVER () clause.

**Usage**
This function returns a double-precision floating-point value containing the p-value of the lack-of-fit statistic (q) for the time series. TS_LACK_OF_FIT_P calls the function imsls_d_lack_of_fit in the IMSL libraries.

**IMSL mapping**
The arguments of TS_LACK_OF_FIT_P map to the IMSL library function imsls_d_lack_of_fit as follows:

```c
params = imsls_d_arma(n_objs, z[], p, q,
    IMSLS_LEAST_SQUARES,
    IMSLS_CONVERGENCE_TOLERANCE, tolerance,
    IMSL_RESIDUAL, &residual, 0);

correlations = imsls_d_autocorrelation(n_objs-
p+lagmax, residuals, lagmax, 0);

result = imsls_d_lack_of_fit(n_objs, correlations,
    lagmax, npfree, 0);
```

**n_objs** Contains the number of rows in the current window frame.

---

**Note** This function is available only with RAP – The Trading Edition Enterprise.
Alphabetical list of functions

- **z[]** Contains the value of `timeseries_expression` for the current window frame.
- **p** Maps to the `p_value` argument defined in `TS_LACK_OF_FIT_P`.
- **q** Maps to the `q_value` argument defined in `TS_LACK_OF_FIT_P`.
- **lagmax** Maps to the `lagmax` argument defined in `TS_LACK_OF_FIT_P`.
- **npfree** Derived from `p + q`.
- **tolerance** An optional argument using `IMSLS_CONVERGENCE_TOLERANCE`. If null, the IMSL library applies a default value and does not use `IMSLS_CONVERGENCE_TOLERANCE`.

For detailed information on how the IMSL function `imsls_d_lack_of_fit` performs time series calculations, see *IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library*.

### Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

### Example

This example shows an input data table, a SQL statement containing the `TS_LACK_OF_FIT_P` function, and the data values returned by the function. This example uses the following table (called `DATASET`) as its input data. The `DATASET` table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>rownum</td>
<td>data</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>
The following SQL statement returns the \( p \) value of the lack of fit statistic on data from the \( \text{data} \) column:

\[
\text{SELECT TS\_LACK\_OF\_FIT\_P(data,1,1,5,0.225) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET}
\]

Sybase IQ returns 50 rows, each containing the same value:

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.735006</td>
</tr>
<tr>
<td>0.735006</td>
</tr>
<tr>
<td>0.735006</td>
</tr>
<tr>
<td>0.735006</td>
</tr>
<tr>
<td>0.735006</td>
</tr>
<tr>
<td>0.735006</td>
</tr>
<tr>
<td>0.735006</td>
</tr>
<tr>
<td>0.735006</td>
</tr>
<tr>
<td>0.735006</td>
</tr>
<tr>
<td>0.735006</td>
</tr>
</tbody>
</table>

... 0.735006

See also


**TS\_MAX\_ARMA\_AR function [Time Series]**

**Note** This function is available only with RAP – The Trading Edition Enterprise.

**Function**
Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns the requested autoregressive estimate.

**Syntax**

\[
\text{TS\_MAX\_ARMA\_AR (timeseries\_expression, ar\_count, ar\_elem)}
\]

\[
\text{OVER (window\_spec)}
\]
Parameters

- **timeseries_expression**: A numeric expression, generally a column name, containing an element in a time series.
- **ar_count**: An integer containing the number of autoregressive values to compute.
- **ar_elem**: An integer identifying which element in the computed autoregressive array is to be returned. The integer must be greater than 0 and less than or equal to `ar_count`.
- **window-spec**: `TS_MAX_ARMA_AR` is an OLAP function requiring an `OVER()` clause.

Usage

This function returns a double-precision floating-point value containing the autoregressive estimate. `TS_MAX_ARMA_AR` calls the function `imsls_d_max_arma` in the IMSL libraries.

IMSL mapping

The arguments of `TS_MAX_ARMA_AR` map to the IMSL library function `imsls_d_max_arma` as follows:

```c
params = imsls_d_max_arma(n_objs, z[], p, q, 0);
```

- **n_objs**: Contains the number of rows in the current window frame.
- **z[]**: Contains the value of `timeseries_expression` for the current window frame.
- **p**: Maps to the `ar_count` argument.
- **q =1**.

For detailed information on how the IMSL function `imsls_d_max_arma` performs time series calculations, see *IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library*.

Example 1

This example shows an input data table, a SQL statement containing the `TS_MAX_ARMA_AR` function, and the data values returned by the function.

This example uses the following table (called `DATASET`) as its input data. The `DATASET` table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
</tbody>
</table>

Reference: Building Blocks, Tables, and Procedures
### Alphabetical list of functions

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
</tbody>
</table>
The following SQL statement returns the second element from an array containing two autoregressive estimates of data from the data column:

```sql
SELECT TS_MAX_ARMA_AR(data,2,2) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>

The following SQL statement returns the second element from an array containing two autoregressive estimates of data from the data column:

```sql
SELECT TS_MAX_ARMA_AR(data,2,1) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS ar_elem1, TS_MAX_ARMA_AR(data,2,2) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS ar_elem2 FROM DATASET
```

**Table 4-49: Values returned from TS_MAX_ARMA_AR Example 1**

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.179748</td>
</tr>
<tr>
<td>0.179748</td>
</tr>
<tr>
<td>0.179748</td>
</tr>
<tr>
<td>0.179748</td>
</tr>
<tr>
<td>0.179748</td>
</tr>
<tr>
<td>0.179748</td>
</tr>
<tr>
<td>0.179748</td>
</tr>
<tr>
<td>0.179748</td>
</tr>
<tr>
<td>0.179748</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**Example 2**

This example provides a sample query that returns two columns of results from the DATASET table—the first and second elements of the autoregressive estimates. See Table 4-48 on page 305 for the DATASET table.

```sql
SELECT TS_MAX_ARMA_AR(data,2,1) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS ar_elem1, TS_MAX_ARMA_AR(data,2,2) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS ar_elem2 FROM DATASET
```
Sybase IQ returns 50 rows of data, each containing the same two values:

<table>
<thead>
<tr>
<th>ar_elem1</th>
<th>ar_elem2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ar_elem1</th>
<th>ar_elem2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.731164</td>
<td>0.179748</td>
</tr>
</tbody>
</table>

Standards and compatibility

- SQL2008  Sybase extension
- Sybase  Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also


**TS_MAX_ARMA_CONST function [Time Series]**

**Note**  This function is available only with RAP – The Trading Edition Enterprise.

**Function**
Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns the constant estimate.

**Syntax**

```
TS_MAX_ARMA_CONST (timeseries_expression)
OVER (window-spec)
```

**Parameters**

- `timeseries_expression` A numeric expression, generally a column name, containing an element in a time series.
**window-spec**  
TS_MAX_ARMA_CONST is an OLAP function requiring an OVER () clause.

**Usage**  
This function returns a double-precision floating-point value containing the constant estimate. TS_MAX_ARMA_CONST calls the function imsls_d_arma in the IMSL libraries.

**IMSL mapping**  
The arguments of TS_MAX_ARMA_CONST map to the IMSL library function imsls_d_arma as follows:

\[
\text{params} = \text{imsls_d_max_arma}(n\_objs, z, p, q, 0);
\]

- **n_objs**  
  Contains the number of rows in the current window frame.

- **z[]**  
  Contains the value of **timeseries_expression** for the current window frame.

- **p**  
  = 1.

- **q**  
  = 1.

For detailed information on how the IMSL function imsls_d_arma performs time series calculations, see **IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library**.

**Example**  
This example shows an input data table, a SQL statement containing the TS_MAX_ARMA CONST function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
</tbody>
</table>

Reference: Building Blocks, Tables, and Procedures
### Alphabetical list of functions

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>
The following SQL statement returns the constant estimate of the maximum likelihood autoregressive calculation on data from the data column:

```
SELECT TS_MAX_ARMA_CONST(data) OVER (ORDER BY rownum 
   ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED 
   FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0.107555</td>
</tr>
</tbody>
</table>

**Standards and compatibility**

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

**See also**

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*


**TS_MAX_ARMA_LIKELIHOOD function [Time Series]**

**Note** This function is available only with RAP – The Trading Edition Enterprise.
Function

Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns likelihood value (ln) for the fitted model.

Syntax

\[
\text{TS\_MAX\_ARMA\_LIKELIHOOD} (\text{timeseries\_expression})
\text{OVER} (\text{window-spec})
\]

Parameters

- **timeseries_expression** A numeric expression, generally a column name, containing an element in a time series.
- **window-spec** TS\_MAX\_ARMA\_LIKELIHOOD is an OLAP function requiring an OVER () clause.

Usage

This function returns a double-precision floating-point value containing the value of \(-2*(\ln(likelihood))\). TS\_MAX\_ARMA\_LIKELIHOOD calls the function imsls\_d\_max\_arma in the IMSL libraries.

IMSL mapping

The arguments of TS\_MAX\_ARMA\_LIKELIHOOD map to the IMSL library function imsls\_d\_max\_arma as follows:

\[
\text{params} = \text{imsls\_d\_max\_arma}(\text{n\_objs}, \text{z}, \text{p}, \text{q}, \text{IMSL\_LOG\_LIKELIHOOD}, \&\text{likelihood}, 0);
\]

- **n\_objs** Contains the number of rows in the current window frame.
- **z[]** Contains the value of timeseries\_expression for the current window frame.
- **p** = 1.
- **q** = 1.
- **likelihood** Provided by the function call. Contains the log likelihood result.

For detailed information on how the IMSL function imsls\_d\_max\_arma performs time series calculations, see IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library.

Example

This example shows an input data table, a SQL statement containing the TS\_MAX\_ARMA\_LIKELIHOOD function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>rownum</td>
<td>data</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505111</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
</tbody>
</table>
Alphabetical list of functions

The following SQL statement returns the likelihood value of the maximum likelihood estimation on data from the `data` column:

```sql
SELECT TS_MAX_ARMA_LIKELIHOOD(data) OVER (ORDER BY ROWNUM rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>

Table 4-54: Values returned from TS_MAX_ARMA_LIKELIHOOD

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>-11.7818</td>
</tr>
<tr>
<td>-11.7818</td>
</tr>
<tr>
<td>-11.7818</td>
</tr>
<tr>
<td>-11.7818</td>
</tr>
<tr>
<td>-11.7818</td>
</tr>
<tr>
<td>-11.7818</td>
</tr>
<tr>
<td>-11.7818</td>
</tr>
<tr>
<td>-11.7818</td>
</tr>
<tr>
<td>-11.7818</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>-11.7818</td>
</tr>
</tbody>
</table>

Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*
TS_MAX_ARMA_MA function [Time Series]

**Note** This function is available only with RAP – The Trading Edition Enterprise.

**Function**
Calculates the exact maximum likelihood estimation of the parameters in a univariate ARMA (autoregressive moving average) time series model, and returns the requested moving average estimate.

**Syntax**

```sql
TS_MAX_ARMA_MA (timeseries_expression, ma_count, ma_elem)
OVER (window-spec)
```

**Parameters**
- `timeseries_expression` A numeric expression, generally a column name, containing an element in a time series.
- `ma_count` An integer containing the number of auto-regressive values to compute.
- `ma_elem` An integer specifying element in the computed moving average array to return. The integer must be greater than zero, and less than or equal to `ma_count`.
- `window-spec` TS_MAX_ARMA_MA is an OLAP function requiring an OVER () clause.

**Usage**
This function returns double-precision floating-point value containing the autoregressive estimate. TS_MAX_ARMA_MA calls the function `imsls_d_max_arma` in the IMSL libraries.

**IMSL mapping**
The arguments of TS_MAX_ARMA_MA map to the IMSL library function `imsls_d_max_arma` as follows:

```sql
params = imsls_d_max_arma(n_objs, z[], p, q, 0);
```
- `n_objs` Contains the number of rows in the current window frame.
- `z[]` Contains the value of `timeseries_expression` for the current window frame.
- `p` = 1.
- `q` Maps to the TS_MAX_ARMA_MA argument `ma_count`.  

Reference: Building Blocks, Tables, and Procedures
For detailed information on how the IMSL function `imsls_d_max_arma` performs time series calculations, see *IMSL Numerical Library User's Guide: Volume 2 of 2 C Stat Library*.

Example

This example shows an input data table, a SQL statement containing the `TS_MAX_ARMA_MA` function, and the data values returned by the function. This example uses the following table (called `DATASET`) as its input data. The `DATASET` table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
</tbody>
</table>
The following SQL statement returns the moving average of the maximum likelihood estimation on data from the `data` column:

```
SELECT TS_MAX_ARMA_MA(DATASET, 5, 4) OVER (ORDER BY ROWNUM rows BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:
TS_OUTLIER_IDENTIFICATION function [Time Series]

**Note** This function is available only with RAP – The Trading Edition Enterprise.

**Function**
Dectes and determines outliers and simultaneously estimates the model parameters in a time series where the underlying outlier-free series follows a general seasonal or non-seasonal ARMA model.

**Syntax**

```
TS_OUTLIER_IDENTIFICATION (timeseries_expression, p_value, q_value, s_value, d_value, [delta_value[, critical_value]])
```

**Parameters**

- `timeseries_expression` A numeric expression, generally a column name, containing an element in a time series.
**CHAPTER 4   SQL Functions**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_value</code></td>
<td>An integer containing the p-portion of the autoregressive integrated moving average (ARIMA) ((p, 0, q)(0, d, 0)_s) model that the outlier free series follows.</td>
</tr>
<tr>
<td><code>q_value</code></td>
<td>An integer containing the q-portion of the ARIMA ((p, 0, q)(0, d, 0)_s) model that the outlier free series follows.</td>
</tr>
<tr>
<td><code>s_value</code></td>
<td>An integer containing the s-portion of the ARIMA ((p, 0, q)(0, d, 0)_s) model that the outlier free series follows.</td>
</tr>
<tr>
<td><code>d_value</code></td>
<td>An integer containing the d-portion of the ARIMA ((p, 0, q)(0, d, 0)_s) model that the outlier free series follows.</td>
</tr>
<tr>
<td><code>delta_value</code></td>
<td>(Optional) A double precision float value containing the dampening effect parameter used in detecting a temporary change outlier. The integer must be greater than 0 and less than 1. The default value is 0.7.</td>
</tr>
<tr>
<td><code>critical_value</code></td>
<td>(Optional) A double-precision float value used as a threshold for outlier detection. The default is 3.0.</td>
</tr>
</tbody>
</table>

**Usage**

This function returns an outlier-free time series.

`TS_OUTLIER_IDENTIFICATION` calls the function `imsls_d_ts_outlier_identification` in the IMSL libraries.

**IMSL mapping**

The arguments of `TS_OUTLIER_IDENTIFICATION` map to the IMSL library function `imsls_d_ts_outlier_identification` as follows:

```sql
params = imsls_d_ts_outlier_identification(n_objs, model[], z[], 0);
```

- `n_objs` Contains the number of rows in the current window frame.
- `model` An array containing the `TS_OUTLIER_IDENTIFICATION` arguments `p_value`, `s_value`, `q_value`, `d_value`:
  ```sql
  model[0] = p_value;
  model[1] = s_value;
  model[2] = q_value;
  model[3] = d_value;
  ```
- `z[]` Contains the value of `timeseries_expression` for the current window frame.

If `delta_value` is non-null, the arguments of `TS_OUTLIER_IDENTIFICATION` map to the IMSL library function `imsls_d_ts_outlier_identification` as follows:
params = imsls_d_ts_outlier_identification(n_objs, model[], z[], IMSL_DELTA, delta_value, 0);

If *critical_value* is non-null, the arguments of *TS_OUTLIER_IDENTIFICATION* map to the IMSL library function `imsls_d_ts_outlier_identification` as follows:

params = imsls_d_ts_outlier_identification(n_objs, model[], z[], IMSL_CRITICAL, critical_value, 0);

If both *delta_value* and *critical_value* are non-null, the arguments of *TS_OUTLIER_IDENTIFICATION* map to the IMSL library function `imsls_d_ts_outlier_identification` as follows:

params = imsls_d_ts_outlier_identification(n_objs, model[], z[], IMSL_DELTA, delta_value, IMSL_CRITICAL, critical_value, 0);

For detailed information on how the IMSL function `imsls_d_ts_outlier_identification` performs time series calculations, see *IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library*.

This example shows an input data table, a SQL statement containing the *TS_OUTLIER_IDENTIFICATION* function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
</tbody>
</table>
2 The following SQL statement detects and determines outliers on data from the data column:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>

Reference: Building Blocks, Tables, and Procedures
Sybase IQ returns 50 rows:

**Table 4-58: Values returned from TS_OUTLIER_IDENTIFICATION**

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.315523</td>
</tr>
<tr>
<td>0.485859</td>
</tr>
<tr>
<td>0.676886</td>
</tr>
<tr>
<td>1.97381</td>
</tr>
<tr>
<td>2.77555</td>
</tr>
<tr>
<td>2.73657</td>
</tr>
<tr>
<td>2.64233</td>
</tr>
<tr>
<td>4.26118</td>
</tr>
<tr>
<td>3.13641</td>
</tr>
<tr>
<td>4.16566</td>
</tr>
<tr>
<td>2.95952</td>
</tr>
<tr>
<td>2.14504</td>
</tr>
<tr>
<td>1.98799</td>
</tr>
<tr>
<td>0.805859</td>
</tr>
<tr>
<td>0.833405</td>
</tr>
<tr>
<td>2.29075</td>
</tr>
<tr>
<td>1.30045</td>
</tr>
<tr>
<td>0.467122</td>
</tr>
<tr>
<td>-0.170107</td>
</tr>
<tr>
<td>-0.256657</td>
</tr>
<tr>
<td>-0.382597</td>
</tr>
<tr>
<td>-0.505511</td>
</tr>
<tr>
<td>-1.90147</td>
</tr>
<tr>
<td>-0.981688</td>
</tr>
<tr>
<td>-1.43116</td>
</tr>
<tr>
<td>-1.39389</td>
</tr>
<tr>
<td>-2.34823</td>
</tr>
<tr>
<td>-2.91122</td>
</tr>
<tr>
<td>-0.927423</td>
</tr>
<tr>
<td>-0.044383</td>
</tr>
<tr>
<td>-0.389648</td>
</tr>
</tbody>
</table>
Standards and compatibility

- **SQL2008** Sybase extension
- **Sybase** Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*


**TS_PARTIAL_AUTOCORRELATION function [Time Series]**

Note: This function is available only with RAP – The Trading Edition Enterprise.

Function: Calculates the sample partial autocorrelation function of a stationary time series.
TS_PARTIAL_AUTOCORRELATION (timeseries_expression, lagmax, lag_elem) OVER (window-spec)

Parameters

timeseries_expression  A numeric expression, generally a column name, containing an element in a time series.

lagmax  An integer containing the maximum lag of autocovariance, autocorrelations, and standard errors of autocorrelations to be calculated. The integer must be greater than or equal to 1, and less than the number of elements in the time series.

lag_elem  An integer identifying the element in the autocorrelation array to return. The integer must be greater than 0 and less than or equal to lagmax.

window-spec  TS_PARTIAL_AUTOCORRELATION is an OLAP function requiring an OVER () clause.

Usage

This function returns an outlier-free time series. TS_PARTIAL_AUTOCORRELATION calls the function imsls_d_autocorrelation and imsls_d_partial_autocorrelation in the IMSL libraries.

IMSL mapping

The arguments of TS_PARTIAL_AUTOCORRELATION map to the IMSL library functions imsls_d_autocorrelation and imsls_d_partial_autocorrelation as follows:

\[
\text{params} = \text{imsls}_d\_\text{autocorrelation}(n\_\text{objs}, z[], \text{lagmax}, 0);
\]

\[
\text{result} = \text{imsls}_d\_\text{partial\_autocorrelation}(\text{lagmax}, \text{params}, 0);
\]

n_objx  Contains the number of rows in the current window frame.

z[]  Contains the value of timeseries_expression for the current window frame.

lagmax  Maps to the TS_PARTIAL_AUTOCORRELATION argument lagmax. For detailed information on how the IMSL functions imsls_d_autocorrelation and imsls_d_partial_autocorrelation perform time series calculations, see IMSL Numerical Library User’s Guide: Volume 2 of 2 C Stat Library.

Example

This example shows an input data table, a SQL statement containing the TS_PARTIAL_AUTOCORRELATION function, and the data values returned by the function. This example uses the following table (called DATASET) as its input data. The DATASET table contains 50 rows of time series data:
<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.315523</td>
</tr>
<tr>
<td>2</td>
<td>0.485859</td>
</tr>
<tr>
<td>3</td>
<td>0.676886</td>
</tr>
<tr>
<td>4</td>
<td>1.97381</td>
</tr>
<tr>
<td>5</td>
<td>2.77555</td>
</tr>
<tr>
<td>6</td>
<td>2.73657</td>
</tr>
<tr>
<td>7</td>
<td>2.64233</td>
</tr>
<tr>
<td>8</td>
<td>4.26118</td>
</tr>
<tr>
<td>9</td>
<td>3.13641</td>
</tr>
<tr>
<td>10</td>
<td>4.16566</td>
</tr>
<tr>
<td>11</td>
<td>2.95952</td>
</tr>
<tr>
<td>12</td>
<td>2.14504</td>
</tr>
<tr>
<td>13</td>
<td>1.98799</td>
</tr>
<tr>
<td>14</td>
<td>0.805859</td>
</tr>
<tr>
<td>15</td>
<td>0.833405</td>
</tr>
<tr>
<td>16</td>
<td>2.29075</td>
</tr>
<tr>
<td>17</td>
<td>1.30045</td>
</tr>
<tr>
<td>18</td>
<td>0.467122</td>
</tr>
<tr>
<td>19</td>
<td>-0.170107</td>
</tr>
<tr>
<td>20</td>
<td>-0.256657</td>
</tr>
<tr>
<td>21</td>
<td>-0.382597</td>
</tr>
<tr>
<td>22</td>
<td>-0.505511</td>
</tr>
<tr>
<td>23</td>
<td>-1.90147</td>
</tr>
<tr>
<td>24</td>
<td>-0.981688</td>
</tr>
<tr>
<td>25</td>
<td>-1.43116</td>
</tr>
<tr>
<td>26</td>
<td>-1.39389</td>
</tr>
<tr>
<td>27</td>
<td>-2.34823</td>
</tr>
<tr>
<td>28</td>
<td>-2.91122</td>
</tr>
<tr>
<td>29</td>
<td>-0.927423</td>
</tr>
<tr>
<td>30</td>
<td>-0.044383</td>
</tr>
<tr>
<td>31</td>
<td>-0.389648</td>
</tr>
<tr>
<td>32</td>
<td>0.545008</td>
</tr>
<tr>
<td>33</td>
<td>0.614096</td>
</tr>
<tr>
<td>34</td>
<td>0.364668</td>
</tr>
<tr>
<td>35</td>
<td>1.16043</td>
</tr>
<tr>
<td>36</td>
<td>-0.654063</td>
</tr>
</tbody>
</table>
The following SQL statement returns the first element from an array containing partial autocorrelations of data from the data column:

```
SELECT TS_PARTIAL_AUTOCORRELATION(data,1,1) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM DATASET
```

Sybase IQ returns 50 rows, each containing the same value:

<table>
<thead>
<tr>
<th>rownum</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>0.616094</td>
</tr>
<tr>
<td>38</td>
<td>2.00875</td>
</tr>
<tr>
<td>39</td>
<td>1.86696</td>
</tr>
<tr>
<td>40</td>
<td>2.80171</td>
</tr>
<tr>
<td>41</td>
<td>3.78422</td>
</tr>
<tr>
<td>42</td>
<td>4.11499</td>
</tr>
<tr>
<td>43</td>
<td>2.77188</td>
</tr>
<tr>
<td>44</td>
<td>4.00312</td>
</tr>
<tr>
<td>45</td>
<td>4.21298</td>
</tr>
<tr>
<td>46</td>
<td>5.00413</td>
</tr>
<tr>
<td>47</td>
<td>4.74498</td>
</tr>
<tr>
<td>48</td>
<td>4.89621</td>
</tr>
<tr>
<td>49</td>
<td>3.93273</td>
</tr>
<tr>
<td>50</td>
<td>4.31592</td>
</tr>
</tbody>
</table>

...
Standards and compatibility

- **SQL2008**  Sybase extension
- **Sybase**  Not compatible with SQL Anywhere or Adaptive Server Enterprise

See also


**TS_VWAP function [Time Series]**

**Note**  This function is available only with RAP – The Trading Edition Enterprise.

**Function**

VWAP stands for volume-weighted average price. TS_VWAP calculates the ratio of the value traded to the total volume traded over a particular time horizon. VWAP is a measure of the average price of a stock over a defined trading horizon. You can use TS_VWAP as both a simple and an OLAP-style aggregate function.

Unlike the other time series functions, TS_VWAP does not call the IMSL libraries.

**Syntax 1**  

```
TS_VWAP (price_expression, volume_expression)
```

**Syntax 2**  

```
TS_VWAP (price_expression, volume_expression)
OVER (window-spec)
```

**Parameters**

- **price_expression**  A numeric expression specifying the price to be incorporated into a volume-weighted average.
- **volume_expression**  A numeric expression specifying the volume to be used in calculating a volume-weighted average.
- **window-spec**  If used with Syntax 2, TS_VWAP is an OLAP function requiring an OVER () clause.

**Usage**

Sybase IQ calculates TS_VWAP using the following formula:
**Figure 4-1: VWAP calculation**

\[ P_{vwap} = \frac{\sum_j P_j \cdot Q_j}{\sum_j Q_j} \]

Pvwap = volume weighted average price  
\( P_j \) = price of trade \( j \).  
\( Q_j \) = quantity of trade \( j \).  
\( j \) = an individual trade that occurred during the time horizon.

**Example**

This example shows an input data table, a SQL statement containing the TS_VWAP function, and the data values returned by the function. This example uses the following table (called VWAP_DATASET) as its input data. The VWAP_DATASET table contains three rows of time series data:

**Table 4-61: Input data table VWAP_DATASET**

<table>
<thead>
<tr>
<th>rownum</th>
<th>price</th>
<th>volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

The following SQL statement calculates the volume weighted average price:

```sql
SELECT TS_VWAP(price,volume) OVER (ORDER BY rownum ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS res FROM VWAP_DATASET
```

Sybase IQ returns three rows:

**Table 4-62: Values returned from TS_VWAP**

<table>
<thead>
<tr>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
</tr>
<tr>
<td>2.5</td>
</tr>
<tr>
<td>2.5</td>
</tr>
</tbody>
</table>

**Standards and compatibility**

- **SQL2008**  Sybase extension  
- **Sybase**  Not compatible with SQL Anywhere or Adaptive Server Enterprise

**See also**

Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*
CHAPTER 4  SQL Functions

UCASE function [String]

Function
Converts all characters in a string to uppercase.

Syntax
UCASE ( string-expression )

Parameters
string-expression  The string to be converted to uppercase.

Example
The following statement returns the value “CHOCOLATE”:

    SELECT UCASE( 'ChocoLate' ) FROM iq_dummy

Usage
The result data type of a UCASE function is a LONG VARCHAR. If you use
UCASE in a SELECT INTO statement, you must have a Large Objects
Management option license or use CAST and set UCASE to the correct data
type and size.

Standards and compatibility
• SQL92  Vendor extension
• Sybase  UCASE is not supported by Adaptive Server Enterprise, but
          UPPER provides the same feature in a compatible manner

See also
“LCASE function [String]” on page 194
“UPPER function [String]” on page 329

UPPER function [String]

Function
Converts all characters in a string to uppercase.

Syntax
UPPER ( string-expression )

Parameters
string-expression  The string to be converted to uppercase.

Example
The following statement returns the value “CHOCOLATE”:

    SELECT UPPER( 'ChocoLate' ) FROM iq_dummy

Usage
The result data type of an UPPER function is a LONG VARCHAR. If you use
UPPER in a SELECT INTO statement, you must have a Large Objects
Management option license or use CAST and set UPPER to the correct data
type and size.

Standards and compatibility
• SQL92  This function is SQL92 compatible.
• Sybase  Compatible with Adaptive Server Enterprise.

See also
“LCASE function [String]” on page 194

Reference: Building Blocks, Tables, and Procedures  329
USER_ID function [System]

Function
Returns an integer user identification number.

Syntax
USER_ID ([user-name])

Parameters
user-name  The user name.

Examples
The following statement returns the user identification number 1:

SELECT USER_ID ('DBA') FROM iq_dummy

The following statement returns the user identification number 0:

SELECT USER_ID ('SYS') FROM iq_dummy

Standards and compatibility
• SQL92  Vendor extension
• Sybase  Adaptive Server Enterprise function implemented for Sybase IQ

See also
“SUSER_ID function [System]” on page 266
“USER_NAME function [System]” on page 330

USER_NAME function [System]

Function
Returns the user name.

Syntax
USER_NAME ([user-id])

Parameters
user-id  The user identification number.

Examples
The following statement returns the value “DBA”:

SELECT USER_NAME (1) FROM iq_dummy

The following statement returns the value “SYS”:

SELECT USER_NAME (0) FROM iq_dummy

Standards and compatibility
• SQL92  Vendor extension
• Sybase  Adaptive Server Enterprise function implemented for Sybase IQ. In Adapter Server Enterprise, USER_NAME returns the user name, not the server user name.

See also
“SUSER_NAME function [System]” on page 266
**UUIDTOSTR function [String]**

**Function**
Converts a unique identifier value (UUID, also known as GUID) to a string value.

**Syntax**
```sql
UUIDTOSTR ( uuid-expression )
```

**Parameters**
- `uuid-expression` A unique identifier value.

**Example**
To convert a unique identifier value into a readable format, execute a query similar to:

```sql
CREATE TABLE T3 (
    pk uniqueidentifier primary key, c1 int);
INSERT INTO T3 (pk, c1) values (0x12345678123456789012123456789012, 1)
SELECT UUIDTOSTR(pk) FROM T3
```

**Usage**
Converts a unique identifier to a string value in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`, where x is a hexadecimal digit. If the binary value is not a valid unique identifier, NULL is returned.

**Standards and compatibility**
- SQL92 Vendor extension
- SQL99 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise.

**See also**
- “NEWID function [Miscellaneous]” on page 208
- “STRTOUUID function [String]” on page 263
- UNIQUEIDENTIFIER in “Binary data types” on page 77

**VAR_POP function [Aggregate]**

**Function**
Computes the statistical variance of a population consisting of a numeric-expression, as a DOUBLE.

**Syntax**
```sql
VAR_POP ( [ ALL ] expression )
```

**Parameters**
- `expression` The expression (commonly a column name) whose population-based variance is calculated over a set of rows.

**Examples**
The following statement lists the average and variance in the number of items per order in different time periods:
Alphabetical list of functions

SELECT year(ShipDate) AS Year, quarter(ShipDate) AS Quarter, AVG(Quantity) AS Average, VAR_POP(Quantity) AS Variance FROM SalesOrderItems GROUP BY Year, Quarter ORDER BY Year, Quarter

Year | Quarter | Average | Variance
---|---|---|---
2000 | 1 | 25.775148 | 203.9021
2000 | 2 | 27.050847 | 225.8109
...

Usage

Computes the population variance of the provided value expression evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of value expression, from the mean of value expression, divided by the number of rows (remaining) in the group or partition.

Population-based variances are computed according to the following formula:

$$\sum_{i=1}^{n} (x_i - \bar{x})^2$$

Standards and compatibility

- SQL99 SQL/foundation feature outside of core SQL
- SQL92 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise

See also

“Analytical functions” on page 104
Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

VAR_SAMP function [Aggregate]

Function

Computes the statistical variance of a sample consisting of a numeric-expression, as a DOUBLE.

Note

VAR_SAMP is an alias of VARIANCE.

Syntax

```
VAR_SAMP ([ ALL ] expression)
```

Parameters

- **expression** The expression (commonly a column name) whose sample-based variance is calculated over a set of rows.
Examples

The following statement lists the average and variance in the number of items per order in different time periods:

```
SELECT year(ShipDate) AS Year, quarter(ShipDate) AS Quarter,
       AVG(Quantity) AS Average,
       VAR_SAMP(Quantity) AS Variance
FROM SalesOrderItems GROUP BY Year, Quarter
ORDER BY Year, Quarter
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Average</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1</td>
<td>25.775148</td>
<td>205.1158</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>27.050847</td>
<td>227.0939</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Usage

Computes the sample variance of value expression evaluated for each row of the group or partition (if DISTINCT was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of value expression, from the mean of value expression, divided by one less than the number of rows (remaining) in the group or partition.

NULL returns NULL for a one-element input set in IQ 12.7 and later. In versions earlier than 12.7, NULL returned zero.

Variances are computed according to the following formula, which assumes a normal distribution:

\[
\sum_{i=1}^{n} (x_i - \bar{x})^2
\]

Standards and compatibility

- **SQL99** SQL/foundation feature outside of core SQL
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also

“Analytical functions” on page 104
“VARIANCE function [Aggregate]” on page 333

**VARIANCE function [Aggregate]**

**Function**

Returns the variance of a set of numbers.

**Syntax**

```
VARIANCE ( [ ALL ] expression )
```
Alphabetical list of functions

Parameters

expression  Any numeric data type (FLOAT, REAL, or DOUBLE) expression.

Examples

Given this data:

```
SELECT Salary FROM Employees WHERE DepartmentID = 300
```

```
salary
51432.000
57090.000
42300.000
43700.00
36500.000
138948.000
31200.000
58930.00
75400.00
```

The following statement returns the value 1063923790.99999994:

```
SELECT VARIANCE (Salary) FROM Employees
WHERE DepartmentID = 300
```

Given this data:

```
SELECT UnitPrice FROM Products WHERE name = 'Tee Shirt'
```

```
UnitPrice
9.00
14.00
14.00
```

The following statement returns the value 8.33333333333334327:

```
SELECT VARIANCE (UnitPrice) FROM Products
WHERE name = 'Tee Shirt'
```

Usage

The formula used to calculate VARIANCE is

\[
\text{var} = \frac{n \sum x^2 - (\sum x)^2}{n(n-1)}
\]

VARIANCE returns a result of data type double-precision floating-point. If applied to the empty set, the result is NULL, which returns NULL for a one-element input set.
VARIANCE does not support the keyword DISTINCT. A syntax error is returned if DISTINCT is used with VARIANCE.

Standards and compatibility
- **SQL92** Vendor extension
- **Sybase** Not supported by Adaptive Server Enterprise

See also
- “STDDEV function [Aggregate]” on page 257
- “VAR_SAMP function [Aggregate]” on page 332
- Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2*

## WEEKS function [Date and time]

**Function**  
Returns the number of weeks since an arbitrary starting date/time, returns the number of weeks between two specified date/times, or adds the specified integer-expression number of weeks to a date/time.

**Syntax**
```
WEEKS ( datetime-expression |
       datetime-expression, datetime-expression |
       datetime-expression, integer-expression )
```

**Parameters**
- **datetime-expression**  
  A date and time.
- **integer-expression**  
  The number of weeks to be added to the `datetime-expression`. If `integer-expression` is negative, the appropriate number of weeks are subtracted from the date/time value. Hours, minutes, and seconds are ignored. If you supply an integer expression, the `datetime-expression` must be explicitly cast as a DATETIME data type.

For information on casting data types, see “CAST function [Data type conversion]” on page 133.

**Examples**
The following statement returns the value 104278:
```
SELECT WEEKS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 9, to signify the difference between the two dates:
```
SELECT WEEKS( '1999-07-13 06:07:12', '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the timestamp value 1999-06-16 21:05:07.000:
```
SELECT WEEKS( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5 ) FROM iq_dummy
```
Alphabetical list of functions

Usage

Weeks are defined as going from Sunday to Saturday, as they do in a North American calendar. The number returned by the first syntax is often useful for determining if two dates are in the same week.

\[
\text{WEEKS ( invoice_sent )} = \text{WEEKS ( payment_received ) FROM iq_dummy}
\]

In the second syntax, the value of WEEKS is calculated from the number of Sundays between the two dates. Hours, minutes, and seconds are ignored. This function is not affected by the DATE_FIRST_DAY_OF_WEEK option.

Standards and compatibility

• SQL92  Vendor extension
• Sybase  Not supported by Adaptive Server Enterprise

WEIGHTED_AVG function [Aggregate]

Function

Calculates an arithmetically (or linearly) weighted average. A weighted average is an average in which each quantity to be averaged is assigned a weight. Weightings determine the relative importance of each quantity that make up the average.

Syntax

\[
\text{WEIGHTED_AVG ( expression ) OVER ( window-spec )}
\]

window-spec: See the Usage section, below.

Parameters

expression  A numeric expression for which a weighted value is being computed.

Usage

Use the WEIGHTED_AVG function to create a weighted moving average. In a weighted moving average, weights decrease arithmetically over time. Weights decrease from the highest weight for the most recent data points, down to zero.

\[
WMA_M = \frac{np_M + (n - 1)p_{M-1} + \cdots + 2p_{M-n+2} + p_{M-n+1}}{n + (n - 1) + \cdots + 2 + 1}
\]

To exaggerate the weighting, you can average two or more weighted moving averages together, or use an EXP_WEIGHTED_AVG function instead.

You can specify elements of window-spec either in the function syntax (inline), or with a WINDOW clause in the SELECT statement.

window-spec:

• Must contain an ORDER BY specifier.
• Cannot contain FOLLOWING or RANGE specifiers.
• The second argument of the ROW specifier—if provided—must be CURRENT ROW.
• Cannot contain NULL values.
• Cannot contain the DISTINCT specifier.
• UNBOUNDED PRECEDING is supported, but may result in poor performance if used

For information on how to specify the window, see “Analytical functions” on page 104.

Example

The following example returns a weighted average of salaries by department for employees in Florida, with the salary of recently hired employees contributing the most weight to the average:

```
SELECT DepartmentID, Surname, Salary,
WEIGHTED_AVG(Salary) OVER (PARTITION BY DepartmentID
ORDER BY YEAR(StartDate) DESC) as "W_AVG"
FROM Employees
WHERE State IN ('FL') ORDER BY DepartmentID
```

The returned result set is:

```
Table 4-63: WEIGHTED_AVG result set

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>Surname</th>
<th>Salary</th>
<th>W_AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Lull</td>
<td>87900.000</td>
<td>87900.000000</td>
</tr>
<tr>
<td>100</td>
<td>Gowda</td>
<td>59840.000</td>
<td>69193.333333</td>
</tr>
<tr>
<td>200</td>
<td>Sterling</td>
<td>64900.000</td>
<td>64900.000000</td>
</tr>
<tr>
<td>200</td>
<td>Kelly</td>
<td>87500.000</td>
<td>79966.666667</td>
</tr>
<tr>
<td>300</td>
<td>Litton</td>
<td>58930.000</td>
<td>58930.000000</td>
</tr>
<tr>
<td>400</td>
<td>Evans</td>
<td>68940.000</td>
<td>68940.000000</td>
</tr>
<tr>
<td>400</td>
<td>Charlton</td>
<td>28300.000</td>
<td>41846.666667</td>
</tr>
<tr>
<td>400</td>
<td>Francis</td>
<td>53870.000</td>
<td>47858.333333</td>
</tr>
</tbody>
</table>
```

Standards and compatibility

• SQL2008 Vendor extension

**WIDTH_BUCKET function [Numerical]**

Function

For a given expression, the WIDTH_BUCKET function returns the bucket number that the result of this expression will be assigned after it is evaluated.
**Syntax**

`WIDTH_BUCKET ( expression, min_value, max_value, num_buckets )`

**Parameters**

- **expression** is the expression for which the histogram is being created. This expression must evaluate to a numeric or datetime value or to a value that can be implicitly converted to a numeric or datetime value. If `expr` evaluates to null, then the expression returns null.

- **min_value** An expression that resolves to the end points of the acceptable range for `expr`. Must also evaluate to numeric or datetime values and cannot evaluate to null.

- **max_value** An expression that resolves to the end points of the acceptable range for `expr`. Must also evaluate to numeric or datetime values and cannot evaluate to null.

- **num_buckets** Is an expression that resolves to a constant indicating the number of buckets. This expression must evaluate to a positive integer.

**Examples**

The following example creates a ten-bucket histogram on the `credit_limit` column for customers in Massachusetts in the sample table and returns the bucket number (“Credit Group”) for each customer. Customers with credit limits greater than the maximum value are assigned to the overflow bucket, 11:

```sql
select EmployeeID, Surname, Salary,
WIDTH_BUCKET(Salary, 29000, 60000, 4) "Wages" from
Employees where State = 'FL' order by "Wages"
```

<table>
<thead>
<tr>
<th>EMPLOYEEID</th>
<th>SURNAME</th>
<th>SALARY</th>
<th>Wages</th>
</tr>
</thead>
<tbody>
<tr>
<td>888</td>
<td>Charlton</td>
<td>28300.000</td>
<td>0</td>
</tr>
<tr>
<td>1390</td>
<td>Litton</td>
<td>58930.000</td>
<td>4</td>
</tr>
<tr>
<td>207</td>
<td>Francis</td>
<td>53870.000</td>
<td>4</td>
</tr>
<tr>
<td>266</td>
<td>Gowda</td>
<td>59840.000</td>
<td>4</td>
</tr>
<tr>
<td>445</td>
<td>Lull</td>
<td>87900.000</td>
<td>5</td>
</tr>
<tr>
<td>1021</td>
<td>Sterling</td>
<td>64900.000</td>
<td>5</td>
</tr>
<tr>
<td>902</td>
<td>Kelly</td>
<td>87500.000</td>
<td>5</td>
</tr>
<tr>
<td>1576</td>
<td>Evans</td>
<td>68940.000</td>
<td>5</td>
</tr>
</tbody>
</table>

When the bounds are reversed, the buckets are open-closed intervals. For example: `WIDTH_BUCKET (credit_limit, 5000, 0, 5)`. In this example, bucket number 1 is (4000, 5000], bucket number 2 is (3000, 4000], and bucket number 5 is (0, 1000]. The overflow bucket is numbered 0 (5000, +infinity), and the underflow bucket is numbered 6 (-infinity, 0].

**Usage**

You can generate equiwidth histograms with the `WIDTH_BUCKET` function. Equiwidth histograms divide data sets into buckets whose interval size (highest value to lowest value) is equal. The number of rows held by each bucket will vary. A related function, `NTILE`, creates equiheight buckets.
Equiwidth histograms can be generated only for numeric, date or datetime data types; therefore, the first three parameters should be all numeric expressions or all date expressions. Other types of expressions are not allowed. If the first parameter is NULL, the result is NULL. If the second or the third parameter is NULL, an error message is returned, as a NULL value cannot denote any end point (or any point) for a range in a date or numeric value dimension. The last parameter (number of buckets) should be a numeric expression that evaluates to a positive integer value; 0, NULL, or a negative value will result in an error.

Buckets are numbered from 0 to (n+1). Bucket 0 holds the count of values less than the minimum. Bucket(n+1) holds the count of values greater than or equal to the maximum specified value.

Standards and compatibility

- SQL03 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise

See also

“NTILE function [Analytical]” on page 212, which creates equiheight histograms.


### YEAR function [Date and time]

Function

Returns a 4-digit number corresponding to the year of the given date/time.

Syntax

```
YEAR (datetime-expression)
```

Parameters

- `datetime-expression` A date and time.

Example

The following statement returns the value 1998:

```
SELECT YEAR ('1998-07-13 06:07:12') FROM iq_dummy
```

Usage

The `YEAR` function is the same as the first syntax of the `YEARS` function.

Standards and compatibility

- SQL92 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise

See also

“YEARS function [Date and time]” on page 339

### YEARS function [Date and time]

Function

Returns a 4-digit number corresponding to the year of a given date/time, returns the number of years between two specified date/times, or adds the specified integer-expression number of years to a date/time.
Alphabetical list of functions

Syntax

YEARS (datetime-expression  
|   datetime-expression, datetime-expression  
|   datetime-expression, integer-expression )

Parameters

datetime-expression  A date and time.

integer-expression  The number of years to be added to the datetime-expression. If integer-expression is negative, the appropriate number of years are subtracted from the datetime value. If you supply an integer expression, the datetime-expression must be explicitly cast as a DATETIME data type.

For information on casting data types, see “CAST function [Data type conversion]” on page 133.

Examples

The following statement returns the value 1998:

```
SELECT YEARS( '1998-07-13 06:07:12' ) FROM iq_dummy
```

The following statement returns the value 2, to signify the difference between the two dates.

```
SELECT YEARS( '1997-07-13 06:07:12',  
               '1999-09-13 10:07:12' ) FROM iq_dummy
```

The following statement returns the YEARS(cast('1999-05-12 21:05:07' as timestamp), 5) value 2004-05-12 21:05:07.000:

```
SELECT YEARS( CAST( '1999-05-12 21:05:07'  
                   AS TIMESTAMP ), 5 ) FROM iq_dummy
```

Usage

The first syntax of the YEARS function is the same as the YEAR function.

The second syntax returns the number of years from the first date to the second date, calculated from the number of first days of the year between the two dates. The number might be negative. Hours, minutes, and seconds are ignored. For example, the following statement returns 2, which is the number of first days of the year between the specified dates:

```
SELECT YEARS ( '2000-02-24', '2002-02-24' ) FROM iq_dummy
```

The next statement also returns 2, even though the difference between the specified dates is not two full calendar years. The value 2 is the number of first days of the year (in this case January 01, 2001 and January 01, 2002) between the two dates.

```
SELECT YEARS ( '2000-02-24', '2002-02-20' ) FROM iq_dummy
```
The third syntax adds an integer-expression number of years to the given date. If the new date is past the end of the month (such as SELECT YEARS ( CAST ( ‘1992-02-29’ AS TIMESTAMP ), 1 )), the result is set to the last day of the month. If integer-expression is negative, the appropriate number of years is subtracted from the date. Hours, minutes, and seconds are ignored.

Standards and compatibility

- SQL92 Vendor extension
- Sybase Not supported by Adaptive Server Enterprise

See also

“YEAR function [Date and time]” on page 339

YMD function [Date and time]

Function

Returns a date value corresponding to the given year, month, and day of the month.

Syntax

YMD ( integer-expression1, integer-expression2, integer-expression3 )

Parameters

- integer-expression1 The year.
- integer-expression2 The number of the month. If the month is outside the range 1–12, the year is adjusted accordingly.
- integer-expression3 The day number. The day is allowed to be any integer, the date is adjusted accordingly.

Examples

The following statement returns the value 1998-06-12:

SELECT YMD( 1998, 06, 12 ) FROM iq_dummy

If the values are outside their normal range, the date adjusts accordingly. For example, the following statement returns the value 1993-03-01:

SELECT YMD( 1992, 15, 1 ) FROM iq_dummy

The following statement returns the value 1993-02-28:

SELECT YMD ( 1992, 15, 1-1 ) FROM iq_dummy

The following statement returns the value 1992-02-29:

SELECT YMD ( 1992, 3, 1-1 ) FROM iq_dummy

Standards and compatibility

- SQL92 Vendor extension
- Sybase Compatible with Adaptive Server Enterprise
CHAPTER 5

Differences from Other SQL Dialects

About this chapter

Sybase IQ conforms to the ANSI SQL89 standard, but has many additional features that are defined in the IBM DB2 and SAA specifications, as well as in the ANSI SQL92 standard.

This chapter describes those features of Sybase IQ that are not commonly found in other SQL implementations.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sybase IQ features</td>
<td>344</td>
</tr>
</tbody>
</table>
Sybase IQ features

The following Sybase IQ features are not found in many other SQL implementations.

Dates
Sybase IQ has date, time, and timestamp types that include year, month, day, hour, minutes, seconds, and fraction of a second. For insertions or updates to date fields, or comparisons with date fields, a free-format date is supported.

In addition, the following operations are allowed on dates:

- **date + integer**  Add the specified number of days to a date.
- **date - integer**  Subtract the specified number of days from a date.
- **date - date**  Compute the number of days between two dates.
- **date + time**  Make a timestamp out of a date and time.

Also, many functions are provided for manipulating dates and times. See Chapter 4, “SQL Functions” for a description of these.

Integrity
Sybase IQ supports both entity and referential integrity. This has been implemented via the following two extensions to the `CREATE TABLE` and `ALTER TABLE` commands.

```
PRIMARY KEY ( column-name, ... )
[NOT NULL] FOREIGN KEY [role-name]
    [(column-name, ...)]
    REFERENCES table-name [(column-name, ...)]
    [ CHECK ON COMMIT ]
```

The PRIMARY KEY clause declares the primary key for the relation. Adaptive Server IQ will then enforce the uniqueness of the primary key, and ensure that no column in the primary key contains the NULL value.

The FOREIGN KEY clause defines a relationship between this table and another table. This relationship is represented by a column (or columns) in this table which must contain values in the primary key of another table. The system then ensures referential integrity for these columns; whenever these columns are modified or a row is inserted into this table, these columns are checked to ensure that either one or more is NULL or the values match the corresponding columns for some row in the primary key of the other table. For more information, see `CREATE TABLE` statement.

Joins
Sybase IQ allows automatic joins between tables. In addition to the NATURAL and OUTER join operators supported in other implementations, Sybase IQ allows KEY joins between tables based on foreign-key relationships. This reduces the complexity of the WHERE clause when performing joins.
### Updates
Sybase IQ allows more than one table to be referenced by the UPDATE command. Views defined on more than one table can also be updated. Many SQL implementations do not allow updates on joined tables.

### Altering tables
The ALTER TABLE command has been extended. In addition to changes for entity and referential integrity, the following types of alterations are allowed:

- `ADD column data-type`
- `MODIFY column data-type`
- `DELETE column`
- `RENAME new-table-name`
- `RENAME old-column TO new-column`

You can use MODIFY to change the maximum length of a character column, as well as converting from one data type to another. See “ALTER TABLE statement,” in “SQL Statements,” in Reference: Statements and Options.

### Subqueries not always allowed
Unlike SQL Anywhere, Sybase IQ does not allow subqueries to appear wherever expressions are allowed. Sybase IQ supports subqueries only as allowed in the SQL-1989 grammar, plus in the SELECT list of the top level query block or in the SET clause of an UPDATE statement. Sybase IQ does not support SQL Anywhere extensions.

Many SQL implementations allow subqueries only on the right side of a comparison operator. For example, the following command is valid in Sybase IQ but not valid in most other SQL implementations.

```sql
SELECT SurName, BirthDate,
       ( SELECT DepartmentName
          FROM Departments
          WHERE DepartmentID = Employees.EmployeeID
          AND DepartmentID = 200 )
FROM Employees
```

### Additional functions
Sybase IQ supports several functions not in the ANSI SQL definition. See Chapter 4, “SQL Functions” for a full list of available functions.

### Cursors
When using Embedded SQL, cursor positions can be moved arbitrarily on the FETCH statement. Cursors can be moved forward or backward relative to the current position or a given number of records from the beginning or end of the cursor.
Sybase IQ features
CHAPTER 6

Physical Limitations

About this chapter

This chapter describes the limitations on size and number of objects in Sybase IQ databases. For limitations that apply to only one platform, see the platform-specific documentation.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size and number limitations</td>
<td>348</td>
</tr>
</tbody>
</table>
Size and number limitations

Table 6-1 lists the limitations on size and number of objects in a Sybase IQ database. In most cases, computer memory and disk drive are more limiting factors.

<table>
<thead>
<tr>
<th>Item</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalog file size</td>
<td>Maximum is 1TB for all platforms except for Windows systems with FAT 32-file systems, which have a 4GB limit. Windows systems with NTFS support the 1TB maximum. Sybase IQ does not support creating dbspaces on NAS (Network Attached Storage) devices.</td>
</tr>
<tr>
<td>Database name size</td>
<td>250 bytes.</td>
</tr>
<tr>
<td>Database size</td>
<td>Maximum database size approximates the number of files times the file size on a particular platform, depending on the maximum disk configuration. Refer to your operating system documentation for kernel parameters that affect the maximum number of files.</td>
</tr>
<tr>
<td>Dbspace size</td>
<td>Raw: No limit – as large as the device allows. Operating system files: 4TB.</td>
</tr>
<tr>
<td>Field size</td>
<td>255 bytes for BINARY, 32,767 bytes for VARBINARY 32,767 for CHAR, VARCHAR. Up to 512 TB for 128 KB pages or 1 PB for 512 KB pages for LONG BINARY, LONG VARCHAR.</td>
</tr>
<tr>
<td>IQ page size</td>
<td>Must be between 64KB and 512KB.</td>
</tr>
<tr>
<td>Maximum key size</td>
<td>255 bytes for single-column index. 5300 bytes for multicolunn index.</td>
</tr>
<tr>
<td>Maximum length of SQL statement</td>
<td>The maximum length of SQL statements is limited to the amount of memory available for the IQ catalog, and to the size of the catalog stack. If your SQL statements are long, increase the catalog stack size using -gss, and increase catalog memory cache size using -c or a combination of -ch and -cl. When printing the SQL statement in error messages, the text is limited to the IQ catalog page size. To print long commands, you can start the server with an increased -gp setting, although in general Sybase recommends that you use the default of -gp 4096.</td>
</tr>
<tr>
<td>Maximum length of variable-length FILLER column</td>
<td>512 bytes.</td>
</tr>
<tr>
<td>Maximum number of users (connected and concurrent)</td>
<td>1000 on 64-bit platforms AIX, HP, Linux, and Sun Solaris. 200 on 32- and 64-bit platforms on Windows.</td>
</tr>
</tbody>
</table>
### Item | Limitation
---|---
Maximum size of temp extract file | Set by TEMP_EXTRACT_SIZE\texttt{n} option. Platform limits are:  
  - AIX & HP-UX: 0 – 64GB  
  - Sun Solaris: 0 – 512GB  
  - Windows: 0 – 128GB  
  - Linux: 0 – 512GB
Number of columns per table | Sybase IQ supports up to 45,000 columns in a table. You may see performance degradation if you have more than 10,000 columns in a table.
Number of events per database | $2^{31} - 1 = 2\,147\,483\,647$.
Number of files per database | Operating system limit that user can adjust; for example, using \texttt{Nofile}. Typically, 2047 files per database.
Number of indexes | $2^{32} = 4,294,967,295$.
Number of rows per table | Limited by table size, upper limit $2^{48}$.
Number of stored procedures per database | $2^{32} = 4,294,967,295$.
Number of tables or views in a single \texttt{FROM} clause | $16 – 64$, depending on the query, with join optimizer turned on.
Number of tables or views referenced per query | 512.
Number of tables per database | 4,293,918,719.
Number of tables per join index (number of tables that can be joined in one query block) | 32.
Number of tables referenced per transaction | No limit.
Number of UNION branches per query | 512. If each branch has multiple tables in the \texttt{FROM} clause, the limit on tables per query reduces the number of UNION branches allowed.
Number of values in an \texttt{IN} list | 250,000.
Row size | Sybase recommends a limit of half the page size.
Table size | Limited by database size.
Size and number limitations
CHAPTER 7 System Procedures

About this chapter
This chapter documents the system-supplied stored procedures in Sybase IQ databases that you can use to retrieve system information.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>System procedure overview</td>
<td>352</td>
</tr>
<tr>
<td>System stored procedures</td>
<td>353</td>
</tr>
<tr>
<td>Catalog stored procedures</td>
<td>492</td>
</tr>
<tr>
<td>Adaptive Server Enterprise system and catalog procedures</td>
<td>522</td>
</tr>
<tr>
<td>SQL Anywhere supported procedures</td>
<td>525</td>
</tr>
</tbody>
</table>
System procedure overview

Sybase IQ includes the following kinds of system procedures:

- System functions that are implemented as stored procedures.
- Catalog stored procedures, for displaying system information in tabular form.
- Multiplex stored procedures, which include both of the above types of procedures, for multiplex server operations. See “System procedures” in Appendix A, “Multiplex Reference,” in Using Sybase IQ Multiplex.
- Transact-SQL system and catalog procedures. For a list of these system procedures, see “Adaptive Server Enterprise system and catalog procedures” on page 522.

This chapter describes system procedures.

System stored procedures related specifically to Large Object data, including sp_iqsetcompression and sp_iqshowcompression, are described in Chapter 5, “Stored Procedure Support” in Large Objects Management in Sybase IQ.

Syntax rules for stored procedures

Use of parentheses and quotes in stored procedure calls varies, depending on whether you enter the procedure name directly, as you can in Interactive SQL, or invoke it with a CALL statement. Some variations are permitted because the product supports both Sybase IQ SQL and Transact-SQL syntax. If you need Transact-SQL compatibility, be sure to use Transact-SQL syntax.

See Table 7-1 for an explanation of syntax variations.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Syntax type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure_name ('param')</td>
<td>Sybase IQ</td>
<td>Quotes are required if you enclose parameters in parentheses.</td>
</tr>
<tr>
<td>procedure_name 'param'</td>
<td>Sybase IQ</td>
<td>Parentheses are optional if you enclose parameters in quotes.</td>
</tr>
<tr>
<td>procedure_name param</td>
<td>Transact-SQL</td>
<td>If you omit quotes around parameters, you must also omit parentheses.</td>
</tr>
<tr>
<td>procedure_name</td>
<td>Sybase IQ or Transact-SQL</td>
<td>Use this syntax if you run a procedure with no parameters directly in DBSQL, and the procedure has no parameters.</td>
</tr>
</tbody>
</table>
When you use Transact-SQL stored procedures, you must use the Transact-SQL syntax.

**Understanding statistics reported by stored procedures**

Many stored procedures report information on the state of Sybase IQ at the time the procedure executes. This means that you get a snapshot view. For example, a report column that lists space in use by a connection shows only the space in use at the instant the procedure executes, not the maximum space used by that connection.

To monitor Sybase IQ usage over an extended period, use the Sybase IQ monitor, which collects and reports statistics from the time you start the monitor until you stop it, at an interval you specify.

**System stored procedures**

System stored procedures are owned by the user ID dbo. The system procedures in this section carry out System Administrator tasks in the IQ Store.

---

**sa_dependent_views procedure**

**Function** Returns the list of all dependent views for a given table or view.
See “sa_dependent_views system procedure” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures > Alphabetical list of system procedures.*

**sa_verify_password procedure**

**Function** Validates the password of the current user.

See “sa_verify_password system procedure” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures > Alphabetical list of system procedures.*

**sa_get_user_status system procedure**

**Function** Allows you to determine the current status of users.

See “sa_get_user_status system procedure” in the SQL Anywhere documentation at *SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures > Alphabetical list of system procedures.*

**sp_expireallpasswords procedure**

**Function** Causes all user passwords to expire immediately.

**Syntax1**

```sql
call sp_expireallpasswords
```

**Syntax2**

```sql
sp_expireallpasswords
```

**See also**

“sp_iqpassword procedure” on page 441

CREATE USER Statement in Chapter 1, “SQL Statements,” in *Reference: Statements and Options.*

**Permissions**

DBA authority required.

**Examples**

Causes all user passwords to expire immediately:

```sql
call sp_expireallpasswords
```
sp_iqaddlogin procedure

Function
Adds a new Sybase IQ user account to the specified login policy.

Syntax1
```sql
call sp_iqaddlogin ('username_in', 'pwd',
[ 'password_expiry_on_next_login' ] [, 'policy_name' ] )
```

Syntax2
```sql
sp_iqaddlogin 'username_in', 'pwd',
[ 'password_expiry_on_next_login' ] [, 'policy_name' ]
```

Syntax3
```sql
sp_iqaddlogin username_in, pwd, [ password_expiry_on_next_login ] [, policy_name ]
```

Usage
- **username_in**  The user’s login name. Login names must conform to the rules for identifiers.
- **pwd**  The user’s password. Passwords must conform to rules for passwords, that is, they must be valid identifiers.
- **password_expiry_on_next_login**  (Optional) Specifies whether user’s password expires as soon as this user’s login is created. Default setting is OFF (password does not expire).
- **policy_name**  (Optional) Creates the user under the named login policy. If unspecified, user is created under the root login policy.

A `username_in/pwd` created using `sp_iqaddlogin` and set to expire in one day is valid all day tomorrow and invalid on the following day. In other words, a login created today and set to expire in `n` days are not usable once the date changes to the `(n+1)th` day.

Permissions
Requires DBA authority.

See also
- “sp_iqpassword procedure” on page 441

Description
Adds a new Sybase IQ user account, assigns a login policy to the user and adds the user to the ISYSUSER system table. If the user already has a user ID for the database but is not in ISYSUSER, (for example, if the user ID was added using the GRANT CONNECT statement or Sybase Central), `sp_iqaddlogin` adds the user to the table.
If you do not specify a login policy name when calling the procedure, Sybase IQ assigns the user to the root login policy.

**Note** If the maximum number of logins for a login policy is unlimited, then a user belonging to that login policy can have an unlimited number of connections.

The first user login forces a password change and assigns a login policy to the newly created user. Sybase recommends that you use **CREATE USER** to create new users, although, for backward compatibility, **sp_iqaddlogin** is still supported.

**Examples**

These calls add the user rose with a password irk324 under the login policy named expired_password. This example assumes the expired_password login policy already exists.

```
call sp_iqaddlogin('rose', 'irk324', 'ON', 'expired_password')
sp_iqaddlogin 'rose','irk324', 'ON', 'expired_password'
```

**sp_iqbackupdetails procedure**

**Function**

Shows all the dbfiles included in a particular backup.

**Syntax**

```
sp_iqbackupdetails backup_id
```

**Parameters**

- **backup_id** Specifies the backup operation transaction identifier.

**Note** You can obtain the **backup_id** value from the SYSIQBACKUPHISTORY table. Run the following query:

```
select * from sysiqbackuphistory
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**

**sp_iqbackupdetails** returns the following:

**Table 7-2: sp_iqbackupdetails columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backup_id</td>
<td>Identifier for the backup transaction.</td>
</tr>
<tr>
<td>backup_time</td>
<td>Time of the backup.</td>
</tr>
</tbody>
</table>

Sybase IQ
### Reference: Building Blocks, Tables, and Procedures

**Example**

Sample output from `sp_iqbackupdetails`:

| backup_id | backup_time       | backup_type       | selective_type       | depends_on_id | dbspace_id | dbspace_name | dbspace_rwstatus | dbspace_createid | dbspace_alterid | dbspace_online | dbspace_size | dbspace_backup_size | dbfile_id | dbfile_name | dbfile_rwstatus | dbfile_createid | dbfile_alterid | dbfile_size | dbfile_backup_size | dbfile_path |
|-----------|-------------------|-------------------|----------------------|---------------|------------|--------------|------------------|-----------------|----------------|---------------|-------------|----------------|-------------|------------|------------------|-------------|----------------|------------------|----------------|----------------|-------------|----------------|-----------------|----------------|
| 883       | 2008-09-23 13:58:49.0 | Full              | All inclusive        | 0             | 0          | system       | ReadWrite       | 0               | 0              | all inclusive | 2884        | 2884         | 0            | C:\\Documents and Settings\\All Users\\SybaseIQ\\demo\\igdemo.db |

**Column name**
- **backup_type**: Type of backup: “Full,” “Incremental since incremental,” or “Incremental since full.”
- **selective_type**: Subtype of backup: “All inclusive”, "All RW files in RW dbspaces", "Set of RO dbspace/file."
- **depends_on_id**: Identifier for previous backup that the backup depends on.
- **dbspace_id**: Name of the dbspace from SYSIQBACKUPHISTORYDETAIL. If dbspace name matches the dbspace name in SYSDBSPACE for a given dbspace_id. Otherwise “null.”
- **dbspace_name**: “ReadWrite” or “Read Only.”
- **dbspace_createid**: Dbspace creation transaction identifier.
- **dbspace_alterid**: Alter DBSPACE read-write mode transaction identifier.
- **dbspace_online**: Status. Values are “Online” or “Offline.”
- **dbspace_size**: Size of dbspace, in KB, at time of backup.
- **dbspace_backup_size**: Size of data, in KB, backed up in the dbspace.
- **dbfile_id**: Identifier for the dbfile being backed up.
- **dbfile_name**: The logical file name, if it was not renamed after the backup operation. If renamed, “null.”
- **dbfile_rwstatus**: “ReadWrite” or “Read Only.”
- **dbfile_createid**: Dbfile creation transaction identifier.
- **dbfile_alterid**: Alter DBSPACE alter FILE read-write mode transaction identifier.
- **dbfile_size**: Size of the dbfile, in KB.
- **dbfile_backup_size**: Size of the dbfile backup, in KB.
- **dbfile_path**: The dbfile path from SYSBACKUPDETAIL, if it matches the physical file path (“file_name”) in SYSDBFILE for a given dbspace_id and the dbfile_id. Otherwise “null.”


**sp_iqbackupsummary procedure**

**Function**
Summarizes backup operations performed.

**Syntax**
```
sp_iqbackupsummary [ timestamp or backup_id ]
```

**Parameters**
- `timestamp or backup_id` Specifies the interval for which to report backup operations. If you specify a timestamp or a backup ID, only those records with `backup_time` greater than or equal to the time you enter are returned. If you specify no timestamp, the procedure returns all the backup records in `ISYSIQBACKUPHISTORY`.

**Permissions**
DBA authority required. Users without DBA authority must be granted `EXECUTE` permission to run the stored procedure.

**Description**
`sp_iqbackupsummary` returns the following:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backup_id</td>
<td>Identifier for the backup transaction</td>
</tr>
<tr>
<td>backup_time</td>
<td>Time of the backup</td>
</tr>
<tr>
<td>backup_type</td>
<td>Type of backup: “Full,” “Incremental since incremental,” or “Incremental since full”</td>
</tr>
<tr>
<td>selective_type</td>
<td>Subtype of backup: &quot;All Inclusive&quot;, &quot;All RW files in RW dbspaces&quot;, &quot;Set of RO dbspace/file&quot;</td>
</tr>
<tr>
<td>virtual_type</td>
<td>Type of virtual backup: “Non-virtual,” “Decoupled,” or “Encapsulated”</td>
</tr>
<tr>
<td>depends_on_id</td>
<td>Identifier for backup that the backup depends on</td>
</tr>
<tr>
<td>creator</td>
<td>Creator of the backup</td>
</tr>
<tr>
<td>backup_size</td>
<td>Size, in KB, of the backup</td>
</tr>
<tr>
<td>user_comment</td>
<td>User comment</td>
</tr>
<tr>
<td>backup_command</td>
<td>The backup statement issued (minus the comment)</td>
</tr>
</tbody>
</table>

**Example**
Sample output of `sp_iqbackupsummary`:

<table>
<thead>
<tr>
<th>backup_id</th>
<th>backup_time</th>
<th>backup_type</th>
<th>selective_type</th>
<th>virtual_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>883</td>
<td>2008-09-23 13:58:49.0</td>
<td>Full</td>
<td>All inclusive</td>
<td>Non virtual</td>
</tr>
</tbody>
</table>
sp_iqcheckdb procedure

Function
Checks validity of the current database. Optionally corrects allocation problems for dbspaces or databases.

sp_iqcheckdb reads all storage in the database. On successful completion, the database free list (an internal allocation map) is updated to reflect the true storage allocation for the database. sp_iqcheckdb then generates a report listing the actions it has performed.

If an error is found, sp_iqcheckdb reports the name of the object and the type of error. sp_iqcheckdb does not update the free list if errors are detected.

sp_iqcheckdb also allows you to check the consistency of a specified table, index, index type, or the entire database.

Note sp_iqcheckdb is the user interface to the IQ database consistency checker (DBCC) and is sometimes referred to as DBCC.

Syntax

```
sp_iqcheckdb 'mode target [...] [ resources resource-percent ]'
```

This is the general syntax of sp_iqcheckdb. There are three modes for checking database consistency, and one for resetting allocation maps. The syntax for each mode is listed separately below. If mode and target are not both specified in the parameter string, Sybase IQ returns the error message:

```
At least one mode and target must be specified to DBCC.
```

Parameters

mode:
{ allocation | check | verify } | dropleaks

target:
[ indextype index-type [...] ]
database | database resetclocks |
{ [ indextype index-type [...] ] table table-name |
[ partition partition-name ] [...] |
index index-name | [...] |
dbspace dbspace-name }

Allocation mode
sp_iqcheckdb 'allocation target [ resources resource-percent ]'

Check mode
sp_iqcheckdb 'check target [ resources resource-percent ]'
System stored procedures

Verify mode

```
sp_iqcheckdb 'verify target [ resources resource-percent ]'
```

Dropleaks mode

```
sp_iqcheckdb 'dropleaks target [ resources resource-percent ]'
```

Usage

database If the target is a database, all dbspaces must be online.

index-type One of the following index types: FP, CMP, LF, HG, HNG, WD, DATE, TIME, DTTM.

If the specified index-type does not exist in the target, an error message is returned. If multiple index types are specified and the target contains only some of these index types, the existing index types are processed by sp_iqcheckdb.

index-name May contain owner and table qualifiers: [[owner.]table-name.]index-name

If owner is not specified, current user and database owner (dbo) are substituted in that order. If table is not specified, index-name must be unique.

table-name May contain an owner qualifier: [owner.]table-name

If owner is not specified, current user and database owner (dbo) are substituted in that order. table-name cannot be a temporary or pre-join table.

Note If either the table name or the index name contains spaces, enclose the table-name or index-name parameter in double quotation marks:

```
sp_iqcheckdb 'check index "dbo.sstab.i2" resources 75'
```

partition-name The partition-name parameter contains no qualifiers. If it contains spaces, enclose it in double quotation marks.

dbspace-name The dbspace-name parameter contains no qualifiers. If it contains spaces, enclose it in double quotation marks.

The partition filter causes sp_iqcheckdb to examine a subset of the corresponding table's rows that belong to that partition. A partition filter on a table and table target without the partition filter are semantically equivalent when the table has only one partition.

The dbspace target examines a subset of the database's pages that belong to that dbspace. The dbspace must be online. The dbspace and database target are semantically equivalent when the table has only one dbspace.
**resource-percent**  The input parameter `resource-percent` must be an integer greater than zero. The resources percentage allows you to limit the CPU utilization of the database consistency checker by controlling the number of threads with respect to the number of CPUs. If `resource-percent` = 100 (the default value), then one thread is created per CPU. If `resource-percent` > 100, then there are more threads than CPUs, which might increase performance for some machine configurations. The minimum number of threads is one.

**Note**  The `sp_iqcheckdb` parameter string must be enclosed in single quotes and cannot be greater than 255 bytes in length.

Allocation problems can be repaired in dropleaks mode.

| Permissions   |  
|---------------|---
| DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure. |

| Description   |  
|---------------|---
| `sp_iqcheckdb` checks the allocation of every block in the database and saves the information in the current session until the next `sp_iqdbstatistics` procedure is issued. `sp_iqdbstatistics` displays the latest result from the most recent execution of `sp_iqcheckdb`. |

`sp_iqcheckdb` can perform several different functions, depending on the parameters specified. The modes for checking and repairing database consistency are:

**Allocation mode**  Checks allocation with blockmap information for the entire database, a specific index, a specific index type, a specific partition, specific table, or a specific dbspace. Does not check index consistency.

Detects duplicate blocks (blocks for which two or more objects claim ownership) or extra blocks (unallocated blocks owned by an object).

Detects leaked blocks (allocated blocks unclaimed by any object in the specified target) for database or dbspace targets.

When the target is a partitioned table, allocation mode:

- Checks metadata of all the table’s partition allocation bitmaps
- Checks metadata of the tables allocation bitmap
- Verifies that blockmap entries are consistent with the table’s allocation bitmap
- Verifies that none of the table’s partition allocation bitmaps overlap
- Checks that rows defined in the table’s partition allocation bitmaps form a superset of the table’s existence bitmap
• Checks that rows defined in the table’s partition allocation bitmaps form a superset of the table’s allocation bitmap

**Note** sp_iqcheckdb cannot check all allocation problems if you specify the name of a single index, index type, or table in the input parameter string.

Run in allocation mode:

• To detect duplicate or unowned blocks (use database or specific tables or indexes as the target)
• If you encounter page header errors

The DBCC option resetclocks is used only with allocation mode. resetclocks is used with forced recovery to convert a multiplex secondary server to a coordinator. For information on multiplex capability, see *Using Sybase IQ Multiplex*. resetclocks corrects the values of internal database versioning clocks, in the event that these clocks are behind. Do not use the resetclocks option for any other purpose, unless you contact Sybase IQ Technical Support.

The resetclocks option must be run in single-user mode and is allowed only with the DBCC statement **allocation database**. The syntax of the resetclocks command is:

```
sp_iqcheckdb 'allocation database resetclocks'
```

**Check mode** Verifies that all database pages can be read for the entire database, specific index, specific index type, specific table, specific partition, or specific dbspace. If the table is partitioned, then check mode will check the table’s partition allocation bitmaps.

Run in check mode if metadata, null count, or distinct count errors are returned when running a query.

**Verify mode** Verifies the contents of non-FP indexes with their corresponding FP indexes for the entire database, a specific index, a specific index type, specific table, specific partition, or specific dbspace. If the specified target contains all data pages for the FP and corresponding non-FP indexes, then verify mode detects the following inconsistencies:

• Missing key – a key that exists in the FP but not in the non-FP index.
• Extra key – a key that exists in the non-FP index but not in the FP index.
• Missing row – a row that exists in the FP but not in the non-FP index.
• Extra row – a row that exists in the non-FP index but not in the FP index.
If the specified target contains only a subset of the FP pages, then verify mode can detect only the following inconsistencies:

- Missing key
- Missing row

If the target is a partitioned table, then verify mode also verifies that each row in the table or table partition has been assigned to the correct partition.

Run in verify mode if metadata, null count, or distinct count errors are returned when running a query.

**Note** `sp_iqcheckdb` does not check referential integrity or repair referential integrity violations.

**Dropleaks mode** When the Sybase IQ server runs in single-node mode, you can use dropleaks mode with either a database or dbspace target to reset the allocation map for the entire database or specified dbspace targets. If the target is a dbspace, then the dropleaks operation must also prevent read-write operations on the named dbspace. All dbspaces in the database or dbspace list must be online.

For information on running dropleaks mode on a multiplex, see *Using Sybase IQ Multiplex*.

The following examples illustrate the use of the `sp_iqcheckdb` procedure.

**Example 1**
In this example, `sp_iqcheckdb` checks the allocation for the entire database:

```sql
sp_iqcheckdb 'allocation database'
```

**Example 2**
In the second example, `sp_iqcheckdb` performs a detailed check on indexes `i1`, `i2`, and `dbo.t1.i3`. If you do not specify a new mode, `sp_iqcheckdb` applies the same mode to the remaining targets, as shown in the following command:

```sql
sp_iqcheckdb 'verify index i1 index i2 index dbo.t1.i3'
```

**Example 3**
You can combine all modes and run multiple checks on a database in a single session. In the following example, `sp_iqcheckdb` performs a quick check of partition `p1` in table `t2`, a detailed check of index `i1`, and allocation checking for the entire database using half of the CPUs:

```sql
sp_iqcheckdb 'check table t2 partition p1 verify index
i1
allocation database resources 50'
```

**Example 4**
This example checks all indexes of the type `FP` in the database:
**System stored procedures**

Example 5

The following example verifies the FP and HG indexes in the table t1 and the LF indexes in the table t2:

```
sp_iqcheckdb 'verify indextype FP index type HG table t1
index type LF table t2'
```

Example 6

The following example illustrates one of the three “LVC cells” messages in the output of sp_iqcheckdb:

```
sp_iqcheckdb 'check index
EFG2JKL.ASIQ_IDX_T208_C504_FP'
```

```
Index Statistics:
** Inconsistent Index:
abcd.EFG2JKL.ASIQ_IDX_T208_C504_FP ****** FP
Indexes Checked: 1
** Unowned LVC Cells: 212 *****
```

The sp_iqcheckdb LVC cells messages include:

- Unowned LVC cells
- Duplicate LVC cell rows
- Unallocated LVC cell rows

These messages indicate inconsistencies with a VARCHAR, VARBINARY, LONG BINARY (BLOB), or LONG VARCHAR (CLOB) column. Unowned LVC cells represent a small amount of unusable disk space and can safely be ignored. Duplicate and Unallocated LVC cells are serious errors that can be resolved only by dropping the damaged columns.

To drop a damaged column, create a new column from a copy of the old column, then drop the original column and rename the new column to the old column.

**Note** LVC is a VARCHAR or VARBINARY column with a width greater than 255. LONG BINARY (BLOB) and LONG VARCHAR (CLOB) also use LVC.

**DBCC performance**

The execution time of DBCC varies, depending on the size of the database for an entire database check, the number of tables or indexes specified, and the size of the machine. Checking only a subset of the database (that is, only specified tables, indexes, or index types) requires less time than checking an entire database.
The processing time of `sp_iqcheckdb dropleaks` mode depends on the number of dbspace targets.

Table 7-4 summarizes the actions and output of the four `sp_iqcheckdb` modes.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Errors detected</th>
<th>Output</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation</td>
<td>Allocation errors</td>
<td>Allocation statistics only</td>
<td>4TB per hour</td>
</tr>
<tr>
<td>Check</td>
<td>Allocation errors</td>
<td>All available statistics</td>
<td>60GB per hour</td>
</tr>
<tr>
<td>Verify</td>
<td>Allocation errors</td>
<td>All available statistics</td>
<td>15GB per hour</td>
</tr>
<tr>
<td>Dropleaks</td>
<td>Allocation errors</td>
<td>Allocation statistics only</td>
<td>4TB per hour</td>
</tr>
</tbody>
</table>

Output

Depending on the execution mode, `sp_iqcheckdb` output includes summary results, errors, informational statistics, and repair statistics. The output may contain as many as three results sets, if you specify multiple modes in a single session. Error statistics are indicated by asterisks (*****), and appear only if errors are detected.

The output of `sp_iqcheckdb` is also copied to the Sybase IQ message file `.iqmsg`. If the DBCC_LOG_PROGRESS option is ON, `sp_iqcheckdb` sends progress messages to the IQ message file, allowing the user to follow the progress of the DBCC operation as it executes.

Output example

The following is an example of the output you see when you run `sp_iqcheckdb 'allocation database'` and there is leaked space. Leaked space is a block that is allocated according to the database free list (an internal allocation map), but DBCC finds that the block is not part of any database object. In this example, DBCC reports 32 leaked blocks.

```
Stat                      Value                      Flags
================================|-----------------------------|-----
DBCC Allocation Mode Report
** DBCC Status              Errors Detected          *****
Allocation Summary
Blocks Total                8192                         |
Blocks in Current Version   4954                         |
Blocks in All Versions      4954                         |
Blocks in Use               4986                         |
```
System stored procedures

<table>
<thead>
<tr>
<th>% Blocks in Use</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>** Blocks Leaked</td>
<td>32</td>
</tr>
<tr>
<td>** 1st Unowned PBN</td>
<td>452</td>
</tr>
<tr>
<td>Total Free Blocks</td>
<td>3206</td>
</tr>
<tr>
<td>Usable Free Blocks</td>
<td>3125</td>
</tr>
<tr>
<td>% Free Space Fragmented</td>
<td>2</td>
</tr>
<tr>
<td>Max Blocks Per Page</td>
<td>16</td>
</tr>
<tr>
<td>1 Block Page Count</td>
<td>97</td>
</tr>
<tr>
<td>3 Block Page Count</td>
<td>153</td>
</tr>
<tr>
<td>4 Block Page Count</td>
<td>14</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>9 Block Hole Count</td>
<td>2</td>
</tr>
<tr>
<td>16 Block Hole Count</td>
<td>194</td>
</tr>
<tr>
<td>Database Objects Checked</td>
<td>1</td>
</tr>
<tr>
<td>B-Array Count</td>
<td>1</td>
</tr>
<tr>
<td>Blockmap Identity Count</td>
<td>1</td>
</tr>
</tbody>
</table>

Allocation Statistics

| Marked Logical Blocks | 8064 |
| Marked Physical Blocks | 4954 |
| Marked Pages | 504 |
| Blocks in Freelist | 126553 |
| Imaginary Blocks | 121567 |
| Highest PBN in Use | 5432 |

** Blocks Leaked | 32 | ****

---

** sp_iqcheckoptions procedure **

Function

For the connected user, sp_iqcheckoptions displays a list of the current value and the default value of database and server startup options that have been changed from the default.

Syntax

sp_iqcheckoptions

Permissions

None. The DBA sees all options set on a permanent basis for all groups and users and sees temporary options set for DBA. Users who are not DBAs see their own temporary options. All users see nondefault server startup options.

Usage

Requires no parameters. Returns one row for each option that has been changed from the default value. The output is sorted by option name, then by user name.

Sybase IQ
For the connected user, the sp_iqcheckoptions stored procedure displays a list of the current value and the default value of database and server startup options that have been changed from the default. sp_iqcheckoptions considers all Sybase IQ and SQL Anywhere database options. Sybase IQ modifies some SQL Anywhere option defaults, and these modified values become the new default values. Unless the new Sybase IQ default value is changed again, sp_iqcheckoptions does not list the option.

When sp_iqcheckoptions is run, the DBA sees all options set on a permanent basis for all groups and users and sees temporary options set for DBA. Users who are not DBAs see their own temporary options. All users see nondefault server startup options.

**Table 7-5: sp_iqcheckoptions columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User_name</td>
<td>The name of the user or group for whom the option has been set. At database creation, all options are set for the public group. Any option that has been set for a group or user other than public is displayed.</td>
</tr>
<tr>
<td>Option_name</td>
<td>The name of the option.</td>
</tr>
<tr>
<td>Current_value</td>
<td>The current value of the option.</td>
</tr>
<tr>
<td>Default_value</td>
<td>The default value of the option.</td>
</tr>
<tr>
<td>Option_type</td>
<td>&quot;Temporary&quot; for a TEMPORARY option, else &quot;Permanent&quot;.</td>
</tr>
</tbody>
</table>

**Examples**

In these examples, the temporary option APPEND_LOAD is set to ON and the group mygroup has the option MAX_WARNINGS set to 9. The user joel has a temporary value of 55 set for MAX_WARNINGS.

In the first example, sp_iqcheckoptions is run by the DBA.

<table>
<thead>
<tr>
<th>User_name</th>
<th>Option_name</th>
<th>Current_value</th>
<th>Default_value</th>
<th>Option_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA</td>
<td>Ansi_update_constr</td>
<td>CURSORS</td>
<td>Off</td>
<td>Permanent</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>Ansi_update_constr</td>
<td>Cursors</td>
<td>Off</td>
<td>Permanent</td>
</tr>
<tr>
<td>DBA</td>
<td>Append_Load</td>
<td>ON</td>
<td>OFF</td>
<td>Temporary</td>
</tr>
<tr>
<td>DBA</td>
<td>Checkpoint_time</td>
<td>20</td>
<td>60</td>
<td>Temporary</td>
</tr>
<tr>
<td>DBA</td>
<td>Connection_authent</td>
<td>Company=MyComp; Application=DBTools;Signa</td>
<td></td>
<td>Temporary</td>
</tr>
<tr>
<td>DBA</td>
<td>Login_procedure</td>
<td>DBA.sp_iq_proce</td>
<td>sp_login_envir</td>
<td>Permanent</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>Login_procedure</td>
<td>DBA.sp_iq_proce</td>
<td>sp_login_envir</td>
<td>Permanent</td>
</tr>
<tr>
<td>mygroup</td>
<td>Max_WARNINGS</td>
<td>9</td>
<td>281474976730655</td>
<td>Permanent</td>
</tr>
<tr>
<td>DBA</td>
<td>Thread_count</td>
<td>25</td>
<td>0</td>
<td>Temporary</td>
</tr>
</tbody>
</table>

In the second example, sp_iqcheckoptions is run by the user joel.
System stored procedures

<table>
<thead>
<tr>
<th>User_name</th>
<th>Option_name</th>
<th>Current_value</th>
<th>Default_value</th>
<th>Option_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>joel</td>
<td>Ansi_update_constr</td>
<td>CURSORS</td>
<td>Off</td>
<td>Permanent</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>Ansi_update_constr</td>
<td>Cursors</td>
<td>Off</td>
<td>Permanent</td>
</tr>
<tr>
<td>joel</td>
<td>Checkpoint_time</td>
<td>20</td>
<td>60</td>
<td>Temporary</td>
</tr>
<tr>
<td>joel</td>
<td>Connection_authent</td>
<td>Company=MyComp;</td>
<td>Application=DBTools;Signa</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temporary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>joel</td>
<td>Login_procedure</td>
<td>DBA.sp_iq_proce sp_login_envir</td>
<td>Permanent</td>
<td></td>
</tr>
<tr>
<td>PUBLIC</td>
<td>Login_procedure</td>
<td>DBA.sp_iq_proce sp_login_envir</td>
<td>Permanent</td>
<td></td>
</tr>
<tr>
<td>joel</td>
<td>Max_Warnings</td>
<td>55</td>
<td>281474976710655</td>
<td>Temporary</td>
</tr>
<tr>
<td>joel</td>
<td>Thread_count</td>
<td>25</td>
<td>0</td>
<td>Temporary</td>
</tr>
</tbody>
</table>

sp_iqclient_lookup procedure

Function

Allows a client application to determine the Sybase IQ user account responsible for a particular data stream, as observed in a network analyzer originating from a specific client IP address/port.

Syntax

sp_iqclient_lookup [ 'IPaddress'], [ Port], [ UserID ]

Parameters

IPaddress Specifies the IP address of the originating client application.

Port Specifies the port number of the originating client application.

UserID Specifies the Sybase IQ user ID.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

The sp_iqclient_lookup procedure takes the client IP address and port number and returns a single row containing Number (the connection ID), IPaddress, Port, and UserID.

1> sp_iqclient_lookup '158.76.235.71',3360
2> go

<table>
<thead>
<tr>
<th>Number</th>
<th>IPaddress</th>
<th>Port</th>
<th>UserID</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>158.76.235.71</td>
<td>3360</td>
<td>rdeniro</td>
</tr>
</tbody>
</table>

Optionally, you can pass a third argument to select only the UserID. If no arguments are passed, sp_iqclient_lookup returns all current logins with their IP addresses and port numbers. For example:

sp_iqclient_lookup

<table>
<thead>
<tr>
<th>Number</th>
<th>IPaddress</th>
<th>Port</th>
<th>UserID</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>162.66.131.36</td>
<td>2082</td>
<td>mbrando</td>
</tr>
</tbody>
</table>
CHAPTER 7  System Procedures

Reference: Building Blocks, Tables, and Procedures

21 162.66.100.233 1863 apacino
22 162.66.100.206 8080 jcaan
23 162.66.100.119 6901 rduvall
24 162.66.100.125 7001 dkeaton
25 162.66.100.124 6347 jcazale

6 rows affected
(return status = 0)

If a client application is not using TCP/IP or for internal connections, the address appears as 127.0.0.1.

Note This information is available for logged on users only. No historical login data is kept on the server for this purpose.

Side effects

The sp_iqclient_lookup stored procedure may impact server performance, which varies from one installation to another. Finding the login name entails scanning through all current active connections on the server; therefore, the impact may be greater on servers with large numbers of connections. Furthermore, this information cannot be cached as it is dynamic — sometimes highly dynamic. It is, therefore, a matter for the local system administrator to manage the use of this stored procedure, as well as monitor the effects on the server, just as for any other client application that uses server facilities.

Examples

Shows IP addresses for UserID jcazale:

    sp_iqclient_lookup null, null, jcazale

<table>
<thead>
<tr>
<th>Number</th>
<th>IPAddress</th>
<th>Port</th>
<th>UserID</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>162.66.131.36</td>
<td>2082</td>
<td>jcazale</td>
</tr>
<tr>
<td>15</td>
<td>164.66.131.36</td>
<td>1078</td>
<td>jcazale</td>
</tr>
</tbody>
</table>

Shows IP addresses from client IP 162.66.131.36:

    sp_iqclient_lookup '162.66.131.36'

<table>
<thead>
<tr>
<th>Number</th>
<th>IPAddress</th>
<th>Port</th>
<th>UserID</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>162.66.131.36</td>
<td>2082</td>
<td>jcazale</td>
</tr>
<tr>
<td>12</td>
<td>162.66.131.36</td>
<td>1078</td>
<td>jcaan</td>
</tr>
</tbody>
</table>

Note The result is empty when the user specifies an incorrect argument.
**sp_iqcolumn procedure**

**Function**
Displays information about columns in a database.

**Syntax1**
```
sp_iqcolumn ([ table_name ], [ table_owner ], [ table_loc ])
```

**Syntax2**
```
sp_iqcolumn
[ table_name='table_name'],[ table_owner='table_owner'],[table_loc='table_loc']
```

**Usage**

**Syntax1** If you specify `table_owner` without specifying `table_name`, you must substitute NULL for `table_name`. For example, `sp_iqcolumn NULL,DBA`.

**Syntax2** The parameters can be specified in any order. Enclose `table_name` and `table_owner` in single quotes.

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Displays information about columns in a database. Specifying the `table_name` parameter returns the columns only from tables with that name. Specifying the `table_owner` parameter returns only tables owned by that user. Specifying both `table_name` and `table_owner` parameters chooses the columns from a unique table, if that table exists. Specifying `table_loc` returns only tables that are defined in that segment type. Specifying no parameters returns all columns for all tables in a database. `sp_iqcolumn` does not return column information for system tables.
### Table 7-6: sp_iqcolumn columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table</td>
</tr>
<tr>
<td>table_owner</td>
<td>The owner of the table</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column</td>
</tr>
<tr>
<td>domain_name</td>
<td>The data type</td>
</tr>
<tr>
<td>width</td>
<td>The precision of numeric data types that have precision and scale or the storage width of numeric data types without scale; the width of character data types</td>
</tr>
<tr>
<td>scale</td>
<td>The scale of numeric data types</td>
</tr>
<tr>
<td>nulls</td>
<td>'Y' if the column can contain NULLS, 'N' if the column cannot contain NULLS</td>
</tr>
<tr>
<td>default</td>
<td>'Identity/Autoincrement' if the column is an identity/autoincrement column, null if not.</td>
</tr>
<tr>
<td>cardinality</td>
<td>The distinct count, if known, by indexes</td>
</tr>
<tr>
<td>est_cardinality</td>
<td>The estimated number of distinct values, set to 255 automatically if the column was created with the MINIMIZE_STORAGE option ON, or a user-supplied value from the IQ UNIQUE constraint specified in CREATE TABLE</td>
</tr>
<tr>
<td>location</td>
<td>TEMP = IQ temporary store, MAIN = IQ main store, SYSTEM = catalog store</td>
</tr>
<tr>
<td>isPartitioned</td>
<td>'Y' if the column belongs to a partitioned table and has one or more partitions whose dbspace is different from the table partition's dbspace, 'N' if the column's table is not partitioned or each partition of the column resides in the same dbspace as the table partition.</td>
</tr>
<tr>
<td>remarks</td>
<td>User comments added with the COMMENT statement</td>
</tr>
<tr>
<td>check</td>
<td>the check constraint expression</td>
</tr>
</tbody>
</table>

#### Example

The following variations in syntax both return all of the columns in the table Departments:

```sql
sp_iqcolumn Departments
call sp_iqcolumn (table_name='Departments')
```
System stored procedures

<table>
<thead>
<tr>
<th>table_name</th>
<th>table_owner</th>
<th>column_name</th>
<th>domain_name</th>
<th>width</th>
<th>scale</th>
<th>nulls</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Departments</td>
<td>GROUPO</td>
<td>DepartmentID</td>
<td>integer</td>
<td>4</td>
<td>0</td>
<td>N</td>
<td>(NULL)</td>
</tr>
<tr>
<td>Departments</td>
<td>GROUPO</td>
<td>DepartmentName</td>
<td>char</td>
<td>40</td>
<td>0</td>
<td>N</td>
<td>(NULL)</td>
</tr>
<tr>
<td>Departments</td>
<td>GROUPO</td>
<td>DepartmentHead</td>
<td>integer</td>
<td>4</td>
<td>0</td>
<td>Y</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

cardinality | est_cardinality | location | isPartitioned | remarks | check |
---|---|---|---|---|---|
5 | 5 | Main | N | (NULL) | (NULL) |
0 | 5 | Main | N | (NULL) | (NULL) |
5 | 5 | Main | N | (NULL) | (NULL) |

The following variation in syntax returns all of the columns in all of the tables owned by table owner DBA.

```
sp_iqcolumn table_owner='DBA'
```

sp_iqcolumnuse procedure

Function
Reports detailed usage information for columns accessed by the workload.

Syntax
```
sp_iqcolumnuse
```

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
Columns from tables created in SYSTEM are not reported.

**Table 7-7: sp_iqcolumnuse columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>Table name</td>
</tr>
<tr>
<td>ColumnName</td>
<td>Column name</td>
</tr>
<tr>
<td>Owner</td>
<td>Username of column owner</td>
</tr>
<tr>
<td>UID**</td>
<td>Column Unique Identifier</td>
</tr>
<tr>
<td>LastDT</td>
<td>Date/time of last access</td>
</tr>
<tr>
<td>NRef</td>
<td>Number of query references</td>
</tr>
</tbody>
</table>

**UID is a number assigned by the system that uniquely identifies the instance of the column (where instance is defined when an object is created).**

Example
Sample output from the sp_iqcolumnuse procedure:

<table>
<thead>
<tr>
<th>TableName</th>
<th>ColumnName</th>
<th>Owner</th>
<th>UID</th>
<th>LastDT</th>
<th>NRef</th>
</tr>
</thead>
<tbody>
<tr>
<td>orders</td>
<td>o_orderdate</td>
<td>DBA</td>
<td>151</td>
<td>20070917 22:41:22..</td>
<td>13</td>
</tr>
<tr>
<td>orders</td>
<td>o_shippriority</td>
<td>DBA</td>
<td>154</td>
<td>20070917 22:41:22..</td>
<td>13</td>
</tr>
<tr>
<td>lineitem</td>
<td>l_orderkey</td>
<td>DBA</td>
<td>186</td>
<td>20070917 22:41:22..</td>
<td>13</td>
</tr>
<tr>
<td>lineitem</td>
<td>l_extendedprice</td>
<td>DBA</td>
<td>191</td>
<td>20070917 22:41:22..</td>
<td>13</td>
</tr>
<tr>
<td>lineitem</td>
<td>l_discount</td>
<td>DBA</td>
<td>192</td>
<td>20070917 22:41:22..</td>
<td>13</td>
</tr>
<tr>
<td>lineitem</td>
<td>l_shipdate</td>
<td>DBA</td>
<td>196</td>
<td>20070917 22:41:22..</td>
<td>13</td>
</tr>
</tbody>
</table>
See also


“INDEX_ADVISOR option” in Chapter 2, “Database Options,” in *Reference: Statements and Options*

---

### sp_iqconnection procedure

**Function**

Shows information about connections and versions, including which users are using temporary dbspace, which users are keeping versions alive, what the connections are doing inside Sybase IQ, connection status, database version status, and so on.

**Syntax**

```
sp_iqconnection [ connhandle ]
```

**Usage**

The input parameter `connhandle` is equal to the *Number* connection property and is the ID number of the connection. The `connection_property` system function returns the connection ID:

```
SELECT connection_property ( 'Number' )
```

When called with an input parameter of a valid `connhandle`, `sp_iqconnection` returns the one row for that connection only.

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.
Description

sp_iqconnection returns a row for each active connection. The columns ConnHandle, Name, Userid, LastReqTime, ReqType, CommLink, NodeAddr, and LastIdle are the connection properties Number, Name, Userid, LastReqTime, ReqType, CommLink, NodeAddr, and LastIdle respectively, and return the same values as the system function sa_conn_info. The additional columns return connection data from the Sybase IQ side of the Sybase IQ engine. Rows are ordered by ConnCreateTime.

The column MPXServerName stores information related to multiplex internode communication (INC), as shown in Table 7-8:

Table 7-8: MPXServerName column values

<table>
<thead>
<tr>
<th>Server where run</th>
<th>MPXServerName column content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplex server</td>
<td>NULL (all connections are local/user connections).</td>
</tr>
</tbody>
</table>
| Multiplex coordinator | • NULL for local/user connections.  
|                   | • Contains value of secondary node’s server name (source of connection) for every INC connection (either on-demand or dedicated heartbeat connection. |
| Multiplex secondary | • NULL for local/user connections.  
|                   | • Contains value of coordinator’s server name (source of connection). |

For information on multiplex capability, see Using Sybase IQ Multiplex.

In Java applications, specify Sybase IQ-specific connection properties from TDS clients in the RemotePWD field. This example, where myconnection becomes the IQ connection name, shows how to specify IQ specific connection parameters:

```java
    p.put("RemotePWD", ",,CON=myconnection");
```

For more details about using the RemotePWD parameter, see “Specifying a database on a server” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - Programming > SQL Anywhere Data Access APIs > SQL Anywhere JDBC driver > Using the jConnect JDBC driver > Supplying a URL to the driver.

Table 7-9: sp_iqconnection columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnHandle</td>
<td>The ID number of the connection.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the server.</td>
</tr>
<tr>
<td>Userid</td>
<td>The user ID for the connection.</td>
</tr>
</tbody>
</table>
### Column name

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastReqTime</td>
<td>The time at which the last request for the specified connection started.</td>
</tr>
<tr>
<td>ReqType</td>
<td>A string for the type of the last request.</td>
</tr>
<tr>
<td>IQCmdType</td>
<td>The current command executing on the Sybase IQ side, if any. The command type reflects commands defined at the implementation level of the engine. These commands consist of transaction commands, DDL and DML commands for data in the IQ store, internal IQ cursor commands, and special control commands such as OPEN and CLOSE DB, BACKUP, RESTORE, and others.</td>
</tr>
<tr>
<td>LastIQCmdTime</td>
<td>The time the last IQ command started or completed on the IQ side of the Sybase IQ engine on this connection.</td>
</tr>
<tr>
<td>IQCursors</td>
<td>The number of cursors open in the IQ store on this connection.</td>
</tr>
<tr>
<td>LowestIQCursorState</td>
<td>The IQ cursor state, if any. If multiple cursors exist on the connection, the state displayed is the lowest cursor state of all the cursors; that is, the furthest from completion. Cursor state reflects internal Sybase IQ implementation detail and is subject to change in the future. For this version, cursor states are: NONE, INITIALIZED, PARSED, DESCRIBED, COSTED, PREPARED, EXECUTED, FETCHING, END_OF_DATA, CLOSED and COMPLETED. As suggested by the names, cursor state changes at the end of the operation. A state of PREPARED, for example, indicates that the cursor is executing.</td>
</tr>
<tr>
<td>IQthreads</td>
<td>The number of Sybase IQ threads currently assigned to the connection. Some threads may be assigned but idle. This column can help you determine which connections are using the most resources.</td>
</tr>
<tr>
<td>TxnID</td>
<td>The transaction ID of the current transaction on the connection. This is the same as the transaction ID displayed in the iqmsg file by the BeginTxn, CmtTxn, and PostCmtTxn messages, as well as the Txn ID Seq logged when the database is opened.</td>
</tr>
<tr>
<td>ConnCreateTime</td>
<td>The time the connection was created.</td>
</tr>
<tr>
<td>TempTableSpaceKB</td>
<td>The number of kilobytes of IQ temporary store space in use by this connection for data stored in IQ temp tables.</td>
</tr>
<tr>
<td>TempWorkSpaceKB</td>
<td>The number of kilobytes of IQ temporary store space in use by this connection for working space such as sorts, hashes, and temporary bitmaps. Space used by bitmaps or other objects that are part of indexes on Sybase IQ temporary tables are reflected in TempTableSpaceKB.</td>
</tr>
<tr>
<td>IQConnID</td>
<td>The 10-digit connection ID displayed as part of all messages in the iqmsg file. This is a monotonically increasing integer unique within a server session.</td>
</tr>
<tr>
<td>satoiq_count</td>
<td>An internal counter used to display the number of crossings from the SQL Anywhere side to the IQ side of the Sybase IQ engine. This might be occasionally useful in determining connection activity. Result sets are returned in buffers of rows and do not increment satoiq_count or iqtosa_count once per row.</td>
</tr>
<tr>
<td>iqtosa_count</td>
<td>An internal counter used to display the number of crossings from the IQ side to the SQL Anywhere side of the Sybase IQ engine. You might find this column to be occasionally useful in determining connection activity.</td>
</tr>
<tr>
<td>CommLink</td>
<td>The communication link for the connection. This is one of the network protocols supported by Sybase IQ, or is local for a same-machine connection.</td>
</tr>
</tbody>
</table>
### System stored procedures

#### Column name | Description
--- | ---
NodeAddr | The node for the client in a client/server connection.
LastIdle | The number of ticks between requests.
MPXServerName | If an INC connection, the varchar(128) value contains the name of the multiplex server where the INC connection originates. NULL if not an INC connection.

### Example

An example of `sp_iqconnection` output:

<table>
<thead>
<tr>
<th>ConnHandle</th>
<th>Name</th>
<th>Userid</th>
<th>LastReqTime</th>
<th>ReqType</th>
<th>IQCmdType</th>
<th>LastIQCmdTime</th>
<th>IQCursors</th>
<th>LowestIQCursorState</th>
<th>IQthreads</th>
<th>TxnID</th>
<th>ConnCreateTime</th>
<th>TempTableSpaceKB</th>
<th>TempWorkSpaceKB</th>
<th>IQconnID</th>
<th>satoiq_count</th>
<th>iqtosa_count</th>
<th>CommLink</th>
<th>NodeAddr</th>
<th>LastIdle</th>
<th>MPXServerName</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>'IQ_MPX_SERVER_H'</td>
<td>'dbo'</td>
<td>'2008-11-18 13:15:00.035'</td>
<td>'EXEC'</td>
<td>'NONE'</td>
<td>2008-11-18</td>
<td>0</td>
<td>'NONE'</td>
<td>1</td>
<td>0</td>
<td>2008-11-18 13:14:09.0</td>
<td>0</td>
<td>0</td>
<td>23198</td>
<td>28</td>
<td>12</td>
<td>'local'</td>
<td>''</td>
<td>2977</td>
<td>'mpx0631_r1'</td>
</tr>
<tr>
<td>11</td>
<td>'IQ_MPX_SERVER_H'</td>
<td>'dbo'</td>
<td>'2008-11-18 13:15:00.046'</td>
<td>'EXEC'</td>
<td>'NONE'</td>
<td>2008-11-18</td>
<td>0</td>
<td>'NONE'</td>
<td>1</td>
<td>0</td>
<td>2008-11-18 13:14:09.0</td>
<td>0</td>
<td>0</td>
<td>23198</td>
<td>28</td>
<td>12</td>
<td>'local'</td>
<td>''</td>
<td>1503</td>
<td>'mpx0631_r2'</td>
</tr>
<tr>
<td>13</td>
<td>'IQ_MPX_SERVER_H'</td>
<td>'dbo'</td>
<td>'2008-11-18 14:52:55.003'</td>
<td>'EXEC'</td>
<td>'NONE'</td>
<td>2008-11-18</td>
<td>0</td>
<td>'NONE'</td>
<td>1</td>
<td>50024</td>
<td>2008-11-18 13:28:08.0</td>
<td>0</td>
<td>0</td>
<td>23207</td>
<td>127</td>
<td>12</td>
<td>'local'</td>
<td>''</td>
<td>10000</td>
<td>'mpx0631_w1'</td>
</tr>
<tr>
<td>15</td>
<td>'IQ_MPX_SERVER_H'</td>
<td>'dbo'</td>
<td>'2008-11-18 14:53:25.005'</td>
<td>'EXEC'</td>
<td>'NONE'</td>
<td>2008-11-18</td>
<td>0</td>
<td>'NONE'</td>
<td>1</td>
<td>50024</td>
<td>2008-11-18 13:28:08.0</td>
<td>0</td>
<td>0</td>
<td>23212</td>
<td>127</td>
<td>12</td>
<td>'local'</td>
<td>''</td>
<td>10000</td>
<td>'mpx0631_w2'</td>
</tr>
<tr>
<td>17</td>
<td>'SQL_DBC_49450e8'</td>
<td>'DBA'</td>
<td>'2008-11-18 14:59:45.680'</td>
<td>'OPEN'</td>
<td>'NONE'</td>
<td>2008-11-18</td>
<td>0</td>
<td>'NONE'</td>
<td>1</td>
<td>50545</td>
<td>2008-11-18 14:03:50.0</td>
<td>0</td>
<td>0</td>
<td>23257</td>
<td>658</td>
<td>65</td>
<td>'TCPIP'</td>
<td>'10.18.60.181'</td>
<td>9375</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>'Sybase Central 1'</td>
<td>'DBA'</td>
<td>'2008-11-18 14:59:45.023'</td>
<td>'CLOSE'</td>
<td>'NONE'</td>
<td>2008-11-18</td>
<td>0</td>
<td>'NONE'</td>
<td>1</td>
<td>50545</td>
<td>2008-11-18 14:03:50.0</td>
<td>0</td>
<td>0</td>
<td>23443</td>
<td>510</td>
<td>54</td>
<td>'local'</td>
<td>''</td>
<td>1238</td>
<td></td>
</tr>
</tbody>
</table>
sp_iqconstraint procedure

Function
Lists referential integrity constraints defined using CREATE TABLE or ALTER TABLE for the specified table or column.

Syntax
sp_iqconstraint ['table-name', 'column-name', 'table-owner']

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
If table name and column name are omitted, reports all referential integrity constraints for all tables including temporary ones in the current connected database. The information includes unique or primary key constraint, referential constraint, and associated role name that are defined by the CREATE TABLE and/or ALTER TABLE statements.

Example
This is sample output that displays all primary key/foreign key pairs where either the candidate key or foreign key contains column ck1 for owner bob in all tables:

call sp_iqconstraint('', 'ck1', 'bob')

PTAB1 bob ASIQ_IDX_T27_HG unique ck1,ck2 selftab bob CK6FK3 Y
ASIQ_IDX_T42_HG ck1,ck2
PTAB2 bob ASIQ_IDX_T27_HG unique ck1,ck2 selftab bob CK6FK4 Y
ASIQ_IDX_T206_I42_HG ck1,ck2
selftab bob ASIQ_IDX_T26_HG unique ck1,ck2 selftab bob CK3FK1 Y
ASIQ_IDX_T206_I42_HG ck1,ck2

The columns displayed are:
• Primary enforced table
• Table owner
• Candidate key index
• Primary key or Unique
• Primary key columns
• Foreign table
• Foreign table owner
• Foreign key role name
• Enforced status (“Y” for enforced, “N” for unenforced)
• Foreign key index
• Foreign key columns
System stored procedures

- Location ("TEMP," "MAIN," or "SYSTEM")

**sp_iqcontext procedure**

**Function**
Tracks and displays, by connection, information about statements that are currently executing.

**Syntax**

```
sp_iqcontext [ connhandle ]
```

**Usage**
The input parameter `connhandle` is equal to the `Number connection` property and is the ID number of the connection.

When called with an input parameter of a valid `connhandle`, `sp_iqcontext` returns the information only for that connection.

**Permissions**
DBA authority required. Users without DBA authority must be granted `EXECUTE` permission to run the stored procedure.

**Description**
sp_iqcontext lets the DBA determine what statements are running on the system at any given moment, and identify the user and connection that issued the statement. With this information, you can use this utility to:

- Match the statement text with the equivalent line in `sp_iqconnection` to get resource usage and transactional information about each connection.
- Match the statement text to the equivalent line in the SQL log created when the `-zr` server option is set to ALL or SQL.
- Use connection information to match the statement text in `sp_iqcontext` to the equivalent line in the `.iqmsg` file, which includes the query plan, when Sybase IQ can collect it.
- Match statement text to an IQ stack trace (`stktrc-yyyyMMdd-hhmmss_#.iq`), if one is produced.
- Collate this information with an operating system stack trace that might be produced, such as `pstack` on Sun Solaris.

The maximum size of statement text collected is the page size of the catalog store.

**Table 7-10: sp_iqcontext columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnOrCursor</td>
<td>CONNECTION or CURSOR.</td>
</tr>
<tr>
<td>ConnHandle</td>
<td>The ID number of the connection.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the server.</td>
</tr>
</tbody>
</table>
### Chapter 7  System Procedures

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Userid</td>
<td>The user ID for the connection or cursor.</td>
</tr>
<tr>
<td>numIQCursors</td>
<td>If column 1 is CONNECTION, the number of cursors open on this connection. If column 1 is CURSOR, a number assigned sequentially to cursors associated with this connection.</td>
</tr>
<tr>
<td>IQthreads</td>
<td>The number of IQ threads currently assigned to the connection. Some threads may be assigned but idle.</td>
</tr>
<tr>
<td>TxnID</td>
<td>The transaction ID of the current transaction.</td>
</tr>
<tr>
<td>ConnOrCurCreateTime</td>
<td>The time this connection or cursor was created.</td>
</tr>
<tr>
<td>IQConnID</td>
<td>The 10-digit connection ID displayed as part of all messages in the .iqmsg file. This is a monotonically increasing integer unique within a server session.</td>
</tr>
<tr>
<td>IQGovernPriority</td>
<td>A value that indicates the order in which the queries of a user are queued for execution. 1 indicates high priority, 2 (the default) medium priority, and 3 low priority. A value of -1 indicates that IQGovernPriority does not apply to the operation. Set the IQGovernPriority value with the database option IQGOVERN_PRIORITY. See “Setting query priority” in Chapter 3, “Optimizing Queries and Deletions” of the Performance and Tuning Guide.</td>
</tr>
<tr>
<td>CmdLine</td>
<td>First 4096 characters of the user command being executed.</td>
</tr>
</tbody>
</table>
System stored procedures

Example

The following example shows an excerpt from output when `sp_iqcontext` is issued with no parameter, producing results for all current connections.

```
CONNECTION 701773517 dba7 DBA 6 1 1324 2009-06-04 09:24:17.000 4 NO COMMAND
CURSOR 701773517 dba7 DBA 1 0 1324 2009-06-04 09:24:46.000 4 2 select * from foo1
CURSOR 701773517 dba7 DBA 2 0 1324 2009-06-04 09:24:47.000 4 2 select a from foo1
... 
CURSOR 701773517 dba7 DBA 6 0 1324 2009-06-04 09:24:47.000 4 2 select e from foo1
```

The first line of output shows connection 701773517 (IQ connection ID 4). This connection is on server dba7, user DBA. It has six active cursors and one IQ thread, and was created from transaction 1324. This connection was not executing a command when `sp_iqcontext` was issued. The next six lines of output list cursors in use by this connection (only three are shown here.)

Two connections are running stored procedures. Connection 1271624950 is running `sp_iqcheckdb` directly from dbisql, has no active cursors but is using 12 IQ threads. Connection 1841476383 has called `sp_iqcontext` as a procedure, is using only 1 IQ thread, and has 10 active cursors (only the first and last are shown here.) In both cases, the name of the stored procedure appears but not the line of code executing within it.

The connection handle (701773517 for the first connection in this example) identifies results in the -zr log. The IQ connection ID (4 for the first connection in this example) identifies results in the .iqmsg file. On UNIX systems, you can use the `grep` command to locate all instances of the connection handle or connection ID, making it easy to correlate information from all sources. The 2 before the user command fragment indicates that this is a medium priority query.

**sp_iqcopyloginpolicy procedure**

Function

Creates a new login policy by copying an existing one.

Syntax1

```
call sp_iqcopyloginpolicy ('existing-policy-name', 'new-policy-name')
```

Syntax2

```
sp_iqcopyloginpolicy 'existing-policy-name', 'new-policy-name'
```

Syntax3

```
sp_iqcopyloginpolicy existing-policy-name, new-policy-name
    policy_name
```

Usage

```
existing policy name
```

The login policy to copy.
new policy name  Name of the new login policy to create (CHAR(128)).

Permissions  Requires DBA authority.

See also  “sp_iqpassword procedure” on page 441

CREATE USER Statement in Chapter 1, “SQL Statements,” in Reference: Statements and Options

Examples  The following stored procedure creates a new login policy named lockeduser by copying the login policy option values from the existing login policy named root.

    call sp_iqcopyloginpolicy ('root','lockeduser')

sp_iqcursorinfo procedure

Function  Displays detailed information about cursors currently open on the server.

Syntax  

sp_iqcursorinfo [ cursor-name ] [, conn-handle ]

Permissions  DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage  

    cursor-name  The name of the cursor. If only this parameter is specified, sp_iqcursorinfo returns information about all cursors that have the specified name in all connections.

    conn-handle  An integer representing the connection ID. If only this parameter is specified, sp_iqcursorinfo returns information about all cursors in the specified connection.

The sp_iqcursorinfo procedure can be invoked without any parameters. If no parameters are specified, sp_iqcursorinfo returns information about all cursors currently open on the server. If both parameters are specified, sp_iqcursorinfo reports information about all of the cursors that have the specified name and are in the specified connection.

If you do not specify the first parameter, but specify the second parameter, you must substitute NULL for the omitted parameter. For example, 

    sp_iqcursorinfo NULL, 1

Reference: Building Blocks, Tables, and Procedures
**System stored procedures**

**Table 7-11: sp_iqcursorinfo usage examples**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_iqcursorinfo</td>
<td>Displays information about all cursors currently open on the server</td>
</tr>
<tr>
<td>sp_iqcursorinfo ‘cursor1’</td>
<td>Displays information about the all cursors named cursor1 in all connections</td>
</tr>
<tr>
<td>sp_iqcursorinfo NULL, 3</td>
<td>Displays information about all cursors in connection 3</td>
</tr>
<tr>
<td>sp_iqcursorinfo ‘cursor2’, 4</td>
<td>Displays information about all the cursors named cursor2 in connection 4</td>
</tr>
</tbody>
</table>

**Description**

The `sp_iqcursorinfo` stored procedure displays detailed information about cursors currently open on the server. The `sp_iqcursorinfo` procedure enables database administrators to monitor cursor status using just one stored procedure and view statistics such as how many rows have been updated, deleted, and inserted.

If you specify one or more parameters, the result is filtered by the specified parameters. For example, if `cursor-name` is specified, only information about the specified cursor is displayed. If `conn-handle` is specified, `sp_iqcursorinfo` returns information only about cursors in the specified connection. If no parameters are specified, `sp_iqcursorinfo` displays information about all cursors currently open on the server.

The `sp_iqcursorinfo` procedure returns information in the following columns:

**Table 7-12: sp_iqcursorinfo columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the cursor</td>
</tr>
<tr>
<td>ConnHandle</td>
<td>The ID number of the connection</td>
</tr>
<tr>
<td>IsUpd</td>
<td>Y: the cursor is updatable; N otherwise</td>
</tr>
<tr>
<td>IsHold</td>
<td>Y: the cursor is a hold cursor; N otherwise</td>
</tr>
<tr>
<td>IQConnID</td>
<td>The ten digit connection ID displayed as part of all messages in the <code>.iqmsg</code> file. This number is a monotonically increasing integer unique within a server session.</td>
</tr>
<tr>
<td>UserID</td>
<td>User ID (or user name) for the user who created and ran the cursor</td>
</tr>
<tr>
<td>CreateTime</td>
<td>The time of cursor creation</td>
</tr>
<tr>
<td>CurrentRow</td>
<td>The current position of the cursor in the result set</td>
</tr>
<tr>
<td>NumFetch</td>
<td>The number of times the cursor fetches a row. The same row can be fetched more than once.</td>
</tr>
<tr>
<td>NumUpdate</td>
<td>The number of times the cursor updates a row, if the cursor is updatable. The same row can be updated more than once.</td>
</tr>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NumDelete</td>
<td>The number of times the cursor deletes a row, if the cursor is updatable.</td>
</tr>
<tr>
<td>NumInsert</td>
<td>The number of times the cursor inserts a row, if the cursor is updatable.</td>
</tr>
<tr>
<td>RWTabOwner</td>
<td>The owner of the table that is opened in RW mode by the cursor.</td>
</tr>
<tr>
<td>RWTabName</td>
<td>The name of the table that is opened in RW mode by the cursor.</td>
</tr>
<tr>
<td>CmdLine</td>
<td>The first 4096 characters of the command the user executed.</td>
</tr>
</tbody>
</table>
**System stored procedures**

**Example**

Display information about all cursors currently open on the server:

```sql
sp_iqcursorinfo
```

<table>
<thead>
<tr>
<th>Name</th>
<th>ConnHandle</th>
<th>IsUpd</th>
<th>IsHold</th>
<th>IQConnID</th>
<th>UserID</th>
</tr>
</thead>
<tbody>
<tr>
<td>crsr1</td>
<td>1</td>
<td>Y</td>
<td>N</td>
<td>118</td>
<td>DBA</td>
</tr>
<tr>
<td>crsr2</td>
<td>3</td>
<td>N</td>
<td>N</td>
<td>118</td>
<td>DBA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CreateTime</th>
<th>CurrentRow</th>
<th>NumFetch</th>
<th>NumUpdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-06-26 15:24:36.000</td>
<td>19</td>
<td>100000000</td>
<td>200000000</td>
</tr>
<tr>
<td>2009-06-26 15:38:38.000</td>
<td>20000</td>
<td>200000000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NumDelete</th>
<th>NumInsert</th>
<th>RWTabOwner</th>
<th>RWTabName</th>
<th>CmdLine</th>
</tr>
</thead>
<tbody>
<tr>
<td>200000000</td>
<td>3000000000</td>
<td>DBA</td>
<td>test1</td>
<td>call proc1()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>call proc2()</td>
</tr>
</tbody>
</table>

**See also**

*Reference: Statements and Options: DECLARE CURSOR statement [ESQL] [SP], DECLARE CURSOR statement [T-SQL], UPDATE (positioned) statement [ESQL] [SP] and DELETE (positioned) statement [ESQL] [SP], FORCE_NO_SCROLL_CURSORS option, and FORCE_UPDATABLE_CURSORS option*

“Using cursors in procedures” in Chapter 1, “Using Procedures and Batches,” in the *System Administration Guide: Volume 2*

“Cursors in transactions” in Chapter 10, “Transactions and Versioning” in the *System Administration Guide: Volume 1*

---

### sp_iqdatatype procedure

**Function**

Displays information about system data types and user-defined data types.

**Syntax**

```sql
sp_iqdatatype [ type-name ], [ type-owner ], [ type-type ]
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**

- **type-name** The name of the data type.
- **type-owner** The name of the creator of the data type.
- **type-type** The type of data type. Allowed values are:
  - **SYSTEM**: displays information about system defined data types (data types owned by user SYS or dbo) only
• **ALL**: displays information about user and system data types
• Any other value: displays information about user data types

The `sp_iqdatatype` procedure can be invoked without any parameters. If no parameters are specified, only information about user-defined data types (data types not owned by `dbo` or `SYS`) is displayed by default.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqdatatype NULL, NULL, SYSTEM` and `sp_iqdatatype NULL, user1`.

### Table 7-13: `sp_iqdatatype` usage examples

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sp_iqdatatype</code></td>
<td>Displays information about all user-defined data types in the database</td>
</tr>
<tr>
<td><code>sp_iqdatatype address</code></td>
<td>Displays information about the user-defined data type named address</td>
</tr>
<tr>
<td><code>sp_iqdatatype non_existing_type</code></td>
<td>No rows returned, as the data type non_existing_type does not exist</td>
</tr>
<tr>
<td><code>sp_iqdatatype NULL, DBA</code></td>
<td>Displays information about all user-defined data types owned by DBA</td>
</tr>
<tr>
<td><code>sp_iqdatatype address, DBA</code></td>
<td>Displays information about the data type address owned by DBA</td>
</tr>
<tr>
<td><code>sp_iqdatatype rowid</code></td>
<td>rowid is a system-defined data type. If there is no user-defined data type also named rowid, no rows are returned. (By default, only user-defined data types are returned.)</td>
</tr>
<tr>
<td><code>sp_iqdatatype rowid, SYS</code></td>
<td>No rows returned, as the data type rowid is not a user-defined data type (by default, only user-defined data types are returned)</td>
</tr>
<tr>
<td><code>sp_iqdatatype NULL, NULL, SYSTEM</code></td>
<td>Displays information about all system defined data types (owned by dbo or SYS)</td>
</tr>
<tr>
<td><code>sp_iqdatatype rowid, NULL, SYSTEM</code></td>
<td>Displays information about the system data type rowid</td>
</tr>
<tr>
<td><code>sp_iqdatatype NULL, NULL, 'ALL'</code></td>
<td>Displays information about the rowid data type</td>
</tr>
</tbody>
</table>

### Description

The `sp_iqdatatype` stored procedure displays information about system and user-defined data types in a database. User-defined data types are also referred to as domains. Predefined domain names are not included in the `sp_iqdatatype` output.
If you specify one or more parameters, the `sp_iqdatatyp` result is filtered by the specified parameters. For example, if `type-name` is specified, only information about the specified data type is displayed. If `type-owner` is specified, `sp_iqdatatyp` only returns information about data types owned by the specified owner. If no parameters are specified, `sp_iqdatatyp` displays information about all the user-defined data types in the database.

The `sp_iqdatatyp` procedure returns information in the following columns:

**Table 7-14: sp_iqdatatyp columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_name</td>
<td>The name of the data type</td>
</tr>
<tr>
<td>creator</td>
<td>The owner of the data type</td>
</tr>
<tr>
<td>nulls</td>
<td>Y indicates the user-defined data type allows nulls; N indicates the data type does not allow nulls.</td>
</tr>
<tr>
<td>width</td>
<td>Displays the length of string columns, the precision of numeric columns, and the number of bytes of storage for all other data types</td>
</tr>
<tr>
<td>scale</td>
<td>Displays the number of digits after the decimal point for numeric data type columns and zero for all other data types</td>
</tr>
<tr>
<td>&quot;default&quot;</td>
<td>The default value for the data type</td>
</tr>
<tr>
<td>&quot;check&quot;</td>
<td>The CHECK condition for the data type</td>
</tr>
</tbody>
</table>

**Example**

Display information about the user-defined data type `address`:

```
sp_iqdatatyp address
```

```
type_name  creator nulls width scale "default" "check"
address     DBA Y 5 0 (NULL) (NULL)
```

See also

CREATE DOMAIN statement in Reference: Statements and Options

Chapter 3, “SQL Data Types”

**sp_iqdbsize procedure**

**Function**

Displays the size of the current database.

**Syntax**

```
sp_iqdbsize(
    [ main ]
)
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.
Description

Returns the total size of the database. Also returns the number of pages required to hold the database in memory and the number of IQ pages when the database is compressed (on disk).

Table 7-15: sp_iqdbsize columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>The path name of the database file.</td>
</tr>
<tr>
<td>Physical Blocks</td>
<td>Total database size in blocks.</td>
</tr>
<tr>
<td>KBytes</td>
<td>Total database size in kilobytes.</td>
</tr>
<tr>
<td>Pages</td>
<td>Total number of IQ pages.</td>
</tr>
<tr>
<td>Compressed Pages</td>
<td>Total number of IQ pages that are compressed (on disk).</td>
</tr>
<tr>
<td></td>
<td>Subset of Pages.</td>
</tr>
<tr>
<td>NBlocks</td>
<td>Total size, in IQ blocks, used to store the data in tables and join indexes.</td>
</tr>
<tr>
<td>Catalog Blocks</td>
<td>Total size, in IQ blocks, used to store the metadata for tables and join indexes. Subset of NBlocks.</td>
</tr>
</tbody>
</table>

Descriptions of sp_iqdbsize columns:

**Database**  The path name of the current database file.

**Physical Blocks** An IQ database consists of one or more dbspaces. Each dbspace has a fixed size, which is originally specified in units of megabytes. This megabyte quantity is converted to blocks using the IQ page size and the corresponding block size for that IQ page size. The Physical Blocks column reflects the cumulative total of each Sybase IQ dbspace size, represented in blocks.

For the correspondence between IQ page size and block size, see Chapter 4, “Managing System Resources” in the Performance and Tuning Guide.

**KBytes** The total size of the database in kilobytes. This value is the total size of the database in blocks (Physical Blocks in the previous sp_iqdbsize column) multiplied by the block size. The block size depends on the IQ page size.

**Pages** The total number of IQ pages necessary to represent in memory all of the data stored in tables and join indexes, as well as the metadata for these objects. This value is always greater than or equal to the value of Compressed Pages (the next sp_iqdbsize column).

**Compressed Pages** The total number of IQ pages necessary to store on disk the data in tables and join indexes as well as the metadata for these objects. This value is always less than or equal to the value of Pages (the previous sp_iqdbsize column), because Sybase IQ compresses pages when the IQ page is written from memory to disk. The sp_iqdbsize Compressed Pages column represents the number of compressed pages.
System stored procedures

NBlocks  The total size in blocks used to store the data in tables and join indexes. This value is always less than or equal to the sp_iqdbsize Physical Blocks value.

Catalog Blocks  The total size in blocks used to store the metadata for tables and join indexes.

Example  Displays size information for the database iqdemo:

```
sp_iqdbsize
```

```
Database
PhysicalBlocks  KBytes  Pages  CompressedPages  NBlocks  CatalogBlocks
=================  ======  =======  ===============  =======  =============
/system1/sybase/IQ-15_1/demo/iqdemo.db
1280  522  688  257  1119  18
```

See also  “Specifying page size” in “Overview of memory use” in Chapter 4, “Managing System Resources” in the Performance and Tuning Guide


sp_iqdbspace procedure

Function  Displays detailed information about each IQ dbspace.

Syntax  
```
sp_iqdbspace [ dbspace-name ]
```

Permissions  DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description  You can use the information from iqdbspace to determine whether data must be moved, and for data that has been moved, whether the old versions have been deallocated. sp_iqdbspace displays this information:
### Table 7-16: sp_iqdbspace columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBSpaceName</td>
<td>Name of the dbspace as specified in the CREATE DBSPACE statement. Dbspace names are case-insensitive for databases created with CASE RESPECT.</td>
</tr>
<tr>
<td>DBSpaceType</td>
<td>Type of the dbspace (MAIN or TEMPORARY only).</td>
</tr>
<tr>
<td>Writable</td>
<td>T (writable) or F (not writable).</td>
</tr>
<tr>
<td>Online</td>
<td>T (online) or F (offline).</td>
</tr>
<tr>
<td>Usage</td>
<td>Percent of dbspace currently in use by all files in the dbspace.</td>
</tr>
<tr>
<td>TotalSize</td>
<td>Total size of all files in the dbspace in the units B (bytes), K (kilobytes), M (megabytes), G (gigabytes), T (terabytes), or P (petabytes).</td>
</tr>
<tr>
<td>Reserve</td>
<td>Total reserved space that can be added to all files in the dbspace.</td>
</tr>
<tr>
<td>NumFiles</td>
<td>Number of files in the dbspace.</td>
</tr>
<tr>
<td>NumRWFiles</td>
<td>Number of read/write files in the dbspace.</td>
</tr>
<tr>
<td>Stripingon</td>
<td>T (On) or F (Off).</td>
</tr>
<tr>
<td>StripeSize</td>
<td>Amount of data written to the dbspace before moving to the next dbspace, if disk striping is on.</td>
</tr>
<tr>
<td>BlkTypes</td>
<td>Space used by both user data and internal system structures (see Table 7-17).</td>
</tr>
<tr>
<td>OkToDrop</td>
<td>‘Y’ indicates the dbspace can be dropped; otherwise ‘N’.</td>
</tr>
</tbody>
</table>

Table 7-17 lists the values of the block type identifiers.
Table 7-17: *sp_iqdbspace block types*

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Block type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Active Version</td>
</tr>
<tr>
<td>B</td>
<td>Backup Structures</td>
</tr>
<tr>
<td>C</td>
<td>Checkpoint Log</td>
</tr>
<tr>
<td>D</td>
<td>Database Identity</td>
</tr>
<tr>
<td>F</td>
<td>Free list</td>
</tr>
<tr>
<td>G</td>
<td>Global Free list Manager</td>
</tr>
<tr>
<td>H</td>
<td>Header Blocks of the free list</td>
</tr>
<tr>
<td>I</td>
<td>Index advice storage</td>
</tr>
<tr>
<td>M</td>
<td>Multiplex CM*</td>
</tr>
<tr>
<td>O</td>
<td>Old Version</td>
</tr>
<tr>
<td>T</td>
<td>Table use</td>
</tr>
<tr>
<td>U</td>
<td>Index use</td>
</tr>
<tr>
<td>N</td>
<td>Column use</td>
</tr>
<tr>
<td>X</td>
<td>Drop at checkpoint</td>
</tr>
</tbody>
</table>

*The multiplex commit identity block (actually 128 blocks) exists in all IQ databases, even though it is not used by simplex databases. For information on multiplex capability, see Using Sybase IQ Multiplex.*

**Example**

The following output displays information about dbspaces.

```
sp_iqdbspace;
```

<table>
<thead>
<tr>
<th>DBSpace Name</th>
<th>DBSpace Type</th>
<th>Writable</th>
<th>Online</th>
<th>Usage</th>
<th>Total Size</th>
<th>Reserve</th>
<th>Num Files</th>
<th>Num RW Files</th>
<th>Stripingon</th>
<th>Stripe Size</th>
<th>Blocks Types</th>
<th>Ok To Drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQ_MAIN</td>
<td>MAIN</td>
<td>T</td>
<td>T</td>
<td>55</td>
<td>75M</td>
<td>200 M</td>
<td>1</td>
<td>1</td>
<td>T</td>
<td>1K</td>
<td>1H, 5169 A, 190 F</td>
<td>N</td>
</tr>
<tr>
<td>IQ_SYSTEM__ MAIN</td>
<td></td>
<td>T</td>
<td>T</td>
<td>21</td>
<td>300M</td>
<td>50M</td>
<td>1</td>
<td>1</td>
<td>F</td>
<td>8K</td>
<td>1H, 7648 F, 32D, 128 M</td>
<td>N</td>
</tr>
<tr>
<td>IQ_SYSTEM__ TEMP</td>
<td>TEMPORARY</td>
<td>T</td>
<td>T</td>
<td>1</td>
<td>100M</td>
<td>50M</td>
<td>1</td>
<td>1</td>
<td>F</td>
<td>8K</td>
<td>1H, 64F, 32A</td>
<td>N</td>
</tr>
</tbody>
</table>

Sybase IQ
See also
“sp_iqdbspaceinfo procedure” on page 391
“sp_iqdbspaceobjectinfo procedure” on page 394
“sp_iqindexinfo procedure” on page 421

sp_iqdbspaceinfo procedure

Function
Displays the size of each object and subobject used in the specified table or join index.

Syntax

```sql
sp_iqdbspaceinfo [ dbspace-name ] [, owner_name ] [, object_name ] [, object-type ]
```

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage
dbspace_name  If specified, sp_iqdbspaceinfo displays one line for each table that has any component in the specified dbspace. Otherwise, the procedure shows information for all dbspaces in the database.

owner_name  Owner of the object. If specified, sp_iqdbspaceinfo displays output only for tables and join indexes with the specified owner. If not specified, sp_iqdbspaceinfo displays information on tables and join indexes for all users in the database.

object_name  Name of the table or join index. If not specified, sp_iqdbspaceinfo displays information on all tables and join indexes in the database.

object_type  Valid object types are table (for table information) or joinindex (for join index information). If unspecified, object type defaults to table.

All parameters are optional and any parameter may be supplied independent of another parameter’s value.

The sp_iqdbspaceinfo stored procedure supports wildcard characters for interpreting dbspace_name, object_name, and owner_name. It shows information for all dbspaces that match the given pattern in the same way the LIKE clause matches patterns inside queries.

Description
sp_iqdbspaceinfo shows the DBA the amount of space used by objects that reside on each dbspace. The DBA can use this information to determine which objects must be relocated before a dbspace can be dropped. The sub-object columns display sizes reported in integer quantities followed by the suffix B, K, M, G, T, or P, representing bytes, kilobytes, megabytes, gigabytes, terabytes, and petabytes, respectively.
For tables, `sp_iqdbspaceinfo` displays sub-object sizing information for all sub-objects (using integer quantities with the suffix B, K, M, G, T, or P). For join indexes, the procedure displays sizing information for the join index and all of its associated sub-objects. The output is sorted by `dbspace_name`, `object_name`, and `owner_name`.

**Table 7-18: sp_iqdbspaceinfo columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbspace_name</td>
<td>Name of the dbspace.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object (table or joinindex only).</td>
</tr>
<tr>
<td>owner</td>
<td>Name of the owner of the object.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object (of type tables and join indexes only) located on the dbspace.</td>
</tr>
<tr>
<td>object_id</td>
<td>Global object ID of the object.</td>
</tr>
<tr>
<td>id</td>
<td>Table id or join-index ID of the object.</td>
</tr>
<tr>
<td>columns</td>
<td>Size of column storage space on the given dbspace.</td>
</tr>
<tr>
<td>indexes</td>
<td>Size of index storage space on the given dbspace. Does not use system-generated indexes (for example, HG indexes in unique constraints or FP indexes).</td>
</tr>
<tr>
<td>metadata</td>
<td>Size of storage space for metadata objects on the given dbspace.</td>
</tr>
<tr>
<td>primary_key</td>
<td>Size of storage space for primary key related objects on the given dbspace.</td>
</tr>
<tr>
<td>unique_constraint</td>
<td>Size of storage space for unique constraint-related objects on the given dbspace.</td>
</tr>
<tr>
<td>foreign_key</td>
<td>Size of storage space for foreign-key-related objects on the given dbspace.</td>
</tr>
</tbody>
</table>

**Note**

If you run `sp_iqdbspaceinfo` against a server you have started with the -r switch (read-only), you see the error Msg 13768, Level 14, State 0: SQL Anywhere Error -757: Modifications not permitted for read-only database. This behavior is expected. The error does not occur on other stored procedures such as `sp_iqdbspace`, `sp_iqfile`, `sp_iqdbspaceobjectinfo` or `sp_iqobjectinfo`.

**Examples**

Displays the size of all objects and sub-objects in all tables in all dbspaces in the database:

```
sp_iqdbspaceinfo

  dbspace_name  object_type owner  object_name  object_id  id  columns
```
### Displaying the size of all objects and sub-objects owned by a specified user in a specified dbspace in the database:

```
sp_iqdbspaceinfo iq_main,GROUPO
```

<table>
<thead>
<tr>
<th>dbspace_name</th>
<th>object_type</th>
<th>owner</th>
<th>object_name</th>
<th>object_id</th>
<th>id</th>
<th>columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Contacts</td>
<td>3538</td>
<td>732</td>
<td>288K</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Customers</td>
<td>3515</td>
<td>731</td>
<td>240K</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Departments</td>
<td>3632</td>
<td>738</td>
<td>72K</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Employees</td>
<td>3641</td>
<td>739</td>
<td>408K</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>FinancialCodes</td>
<td>3612</td>
<td>736</td>
<td>72K</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>FinancialData</td>
<td>3621</td>
<td>737</td>
<td>96K</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Products</td>
<td>3593</td>
<td>735</td>
<td>272K</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>SalesOrderItems</td>
<td>3580</td>
<td>734</td>
<td>120K</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>SalesOrders</td>
<td>3565</td>
<td>733</td>
<td>144K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>indexes</th>
<th>metadata</th>
<th>primary_key</th>
<th>unique_constraint</th>
<th>foreign_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0B</td>
<td>1.37M</td>
<td>0B</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>0B</td>
<td>464K</td>
<td>0B</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>0B</td>
<td>1.22M</td>
<td>0B</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>0B</td>
<td>5.45M</td>
<td>192K</td>
<td>0B</td>
<td>48K</td>
</tr>
<tr>
<td>48K</td>
<td>4.63M</td>
<td>24K</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>0B</td>
<td>1.78M</td>
<td>24K</td>
<td>0B</td>
<td>48K</td>
</tr>
<tr>
<td>0B</td>
<td>8.03M</td>
<td>24K</td>
<td>0B</td>
<td>48K</td>
</tr>
<tr>
<td>0B</td>
<td>1.53M</td>
<td>24K</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>0B</td>
<td>2.19M</td>
<td>24K</td>
<td>0B</td>
<td>48K</td>
</tr>
<tr>
<td>192K</td>
<td>4.67M</td>
<td>24K</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>0B</td>
<td>2.7M</td>
<td>24K</td>
<td>0B</td>
<td>104K</td>
</tr>
<tr>
<td>0B</td>
<td>3.35M</td>
<td>24K</td>
<td>0B</td>
<td>144K</td>
</tr>
</tbody>
</table>
Displays the size of a specified object and its sub-objects owned by a specified user in a specified dbspace in the database:

```
sp_iqdbspaceinfo iq_main,GROUPO,Departments
```

<table>
<thead>
<tr>
<th>dbspace_name</th>
<th>object_type</th>
<th>owner</th>
<th>object_name</th>
<th>object_id</th>
<th>id</th>
<th>columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Departments</td>
<td>3632</td>
<td>738</td>
<td>72K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>indexes</th>
<th>metadata</th>
<th>primary_key</th>
<th>unique_constraint</th>
<th>foreign_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0B</td>
<td>1.78M</td>
<td>24K</td>
<td>0B</td>
<td>48K</td>
</tr>
</tbody>
</table>

See also “sp_iqdbspace procedure” on page 388 and “sp_iqindexinfo procedure” on page 421

**sp_iqdbspaceobjectinfo procedure**

Function

Lists objects of type table and join index and their sub-objects (columns, indexes, metadata, primary keys, unique constraints, foreign keys, and partitions) for a given dbspace.

Syntax

```
sp_iqdbspaceobjectinfo [ dbspace-name ][ , owner_name ][ , object_name ][ , object-type ]
```

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

- **dbspace-name** If specified, `sp_iqdbspaceobjectinfo` displays output only for the specified dbspace. Otherwise, it shows information for all dbspaces in the database.

- **owner-name** Owner of the object. If specified, `sp_iqdbspaceobjectinfo` displays output only for tables and join indexes with the specified owner. If not specified, `sp_iqdbspaceobjectinfo` displays information for tables and join indexes for all users in the database.

- **object-name** Name of the table or join index. If not specified, `sp_iqdbspaceobjectinfo` displays information for all tables and join indexes in the database.

- **object-type** Valid object types are table (for table information) or joinindex (for join index information). If unspecified, object type defaults to table.

All parameters are optional and any parameter may be supplied independent of the value of other parameters.
The `sp_iqdbspaceobjectinfo` stored procedure supports wildcard characters for interpreting `dbspace_name`, `object_name`, and `owner_name`. It displays information for all dbspaces that match the given pattern in the same way as the `LIKE` clause matches patterns inside queries.

**Description**

For tables, `sp_iqdbspaceobjectinfo` displays summary information for all associated sub-objects. For join indexes, it displays sizing information for the join index and for all of its associated sub-objects. The output of the stored procedure is sorted by `dbspace_name`, `owner` and `object_name`.

`sp_iqdbspaceobjectinfo` displays the following information, based on the input parameter values:

**Table 7-19: `sp_iqdbspaceobjectinfo` columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbspace_name</code></td>
<td>Name of the dbspace.</td>
</tr>
<tr>
<td><code>object_type</code></td>
<td>Table or join index.</td>
</tr>
<tr>
<td><code>owner</code></td>
<td>Name of the owner of the object.</td>
</tr>
<tr>
<td><code>object_name</code></td>
<td>Name of the object (of type tables and join indexes only) located on the dbspace.</td>
</tr>
<tr>
<td><code>object_id</code></td>
<td>Global object id of the object.</td>
</tr>
<tr>
<td><code>id</code></td>
<td>Table id or join-index id of the object.</td>
</tr>
<tr>
<td><code>columns</code></td>
<td>Number of table columns which are located on the given dbspace. If a column or one of the column-partitions is located on a dbspace, it is counted to be present on that dbspace. The result is displayed in the form n/N (n out of total N columns of the table are on the given dbspace).</td>
</tr>
<tr>
<td><code>indexes</code></td>
<td>Number of user defined indexes on the table which are located on the given dbspace. Displayed in the form n/N (n out of total N indexes on the table are on the given dbspace). This does not contain indexes which are system generated, such as FP indexes and HG indexes in the case of unique constraints.</td>
</tr>
<tr>
<td><code>metadata</code></td>
<td>Boolean field (Y/N) to denote if the metadata information of the sub-object is also located on this dbspace.</td>
</tr>
<tr>
<td><code>primary_key</code></td>
<td>Boolean field (1/0) to denote if the primary key of the table, if any, is located on this dbspace.</td>
</tr>
<tr>
<td><code>unique_constraint</code></td>
<td>Number of unique constraints on the table which are located on the given dbspace. Displayed in the form n/N (n out of total N unique constraints on the table are in the given dbspace).</td>
</tr>
<tr>
<td><code>foreign_key</code></td>
<td>Number of foreign keys on the table which are located on the given dbspace. Displayed in the form n/N (n out of total N foreign keys on the table are in the given dbspace).</td>
</tr>
</tbody>
</table>
System stored procedures

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitions</td>
<td>Number of partitions of the table which are located on the given dbspace. Displayed in the form n/N (n out of total N partitions of the table are in the given dbspace).</td>
</tr>
</tbody>
</table>

Examples

Displays information about a specific dbspace in the database:

```
sp_iqdbspaceobjectinfo iq_main
```

<table>
<thead>
<tr>
<th>dbspace_name</th>
<th>object_type</th>
<th>owner</th>
<th>object_name</th>
<th>object_id</th>
<th>id</th>
<th>columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>iq_main</td>
<td>table</td>
<td>DBA</td>
<td>empi</td>
<td>3689</td>
<td>741</td>
<td>4/4</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>DBA</td>
<td>iq_dummy</td>
<td>3686</td>
<td>740</td>
<td>1/1</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>DBA</td>
<td>sale</td>
<td>3698</td>
<td>742</td>
<td>4/4</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Contacts</td>
<td>3538</td>
<td>732</td>
<td>12/12</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Customers</td>
<td>3515</td>
<td>731</td>
<td>10/10</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Departments</td>
<td>3632</td>
<td>738</td>
<td>3/3</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Employees</td>
<td>3641</td>
<td>739</td>
<td>21/21</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>FinancialCodes</td>
<td>3612</td>
<td>736</td>
<td>3/3</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>FinancialData</td>
<td>3621</td>
<td>737</td>
<td>4/4</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Products</td>
<td>3593</td>
<td>735</td>
<td>8/8</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>SalesOrderItems</td>
<td>3580</td>
<td>734</td>
<td>5/5</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>SalesOrders</td>
<td>3565</td>
<td>733</td>
<td>6/6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>indexes</th>
<th>metadata</th>
<th>primary_key</th>
<th>unique_constraint</th>
<th>foreign_key</th>
<th>partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/0</td>
<td>Y</td>
<td>0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>0/0</td>
<td>Y</td>
<td>0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>0/0</td>
<td>Y</td>
<td>1</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>1/1</td>
<td>Y</td>
<td>1</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>0/0</td>
<td>Y</td>
<td>1</td>
<td>0/0</td>
<td>1/1</td>
<td>0/0</td>
</tr>
<tr>
<td>0/0</td>
<td>Y</td>
<td>1</td>
<td>0/0</td>
<td>1/1</td>
<td>0/0</td>
</tr>
<tr>
<td>0/0</td>
<td>Y</td>
<td>1</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>0/0</td>
<td>Y</td>
<td>1</td>
<td>0/0</td>
<td>1/1</td>
<td>0/0</td>
</tr>
<tr>
<td>4/4</td>
<td>Y</td>
<td>1</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>0/0</td>
<td>Y</td>
<td>1</td>
<td>0/0</td>
<td>2/2</td>
<td>0/0</td>
</tr>
<tr>
<td>0/0</td>
<td>Y</td>
<td>1</td>
<td>0/0</td>
<td>3/3</td>
<td>0/0</td>
</tr>
</tbody>
</table>

Displays information about the objects owned by a specific user in a specific dbspace in the database:

```
sp_iqdbspaceobjectinfo iq_main,GROUPO
```

<table>
<thead>
<tr>
<th>dbspace_name</th>
<th>object_type</th>
<th>owner</th>
<th>object_name</th>
<th>object_id</th>
<th>id</th>
<th>columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Contacts</td>
<td>3538</td>
<td>732</td>
<td>12/12</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Customers</td>
<td>3515</td>
<td>731</td>
<td>10/10</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Departments</td>
<td>3632</td>
<td>738</td>
<td>3/3</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Employees</td>
<td>3641</td>
<td>739</td>
<td>21/21</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>FinancialCodes</td>
<td>3612</td>
<td>736</td>
<td>3/3</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>FinancialData</td>
<td>3621</td>
<td>737</td>
<td>4/4</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>Products</td>
<td>3593</td>
<td>735</td>
<td>8/8</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>SalesOrderItems</td>
<td>3580</td>
<td>734</td>
<td>5/5</td>
</tr>
<tr>
<td>iq_main</td>
<td>table</td>
<td>GROUPO</td>
<td>SalesOrders</td>
<td>3565</td>
<td>733</td>
<td>6/6</td>
</tr>
</tbody>
</table>
CHAPTER 7 System Procedures

Uses sp_iqdbspaceobjectinfo to construct commands that can be used to move objects. In this example, the commands move all tables on dbspace_x to dbspace_y.

```
SELECT 'ALTER TABLE ' || owner || '.' || object_name || ' MOVE TO dbspace_y;' FROM sp_iqdbspaceobjectinfo() WHERE object_type = 'table' AND dbspace_name = 'dbspace_x';
```

The following ALTER TABLE commands are the result:

```
ALTER TABLE DBA.dt1 MOVE TO dbspace_y;
ALTER TABLE DBA.dt2 MOVE TO dbspace_y;
ALTER TABLE DBA.dt3 MOVE TO dbspace_y;
```

See also “sp_iqdbspace procedure” on page 388 and “sp_iqindexinfo procedure” on page 421

**sp_iqdbstatistics procedure**

**Function**
Reports results of the most recent sp_iqcheckdb.

**Syntax**
```
sp_iqdbstatistics
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Displays the database statistics collected by the most recent execution of sp_iqcheckdb.

**Example**
The following example shows the output from sp_iqdbstatistics. For this example, the most recent execution of sp_iqcheckdb was the command sp_iqcheckdb 'allocation database'.
### System stored procedures

<table>
<thead>
<tr>
<th>DB Statistics</th>
<th>Value</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCC Allocation Mode Report</td>
<td></td>
<td></td>
</tr>
<tr>
<td>** DBCC Status**</td>
<td>Errors Detected</td>
<td>*****</td>
</tr>
<tr>
<td>DBCC Work units Dispatched</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>DBCC Work units Completed</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>** Allocation Summary**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blocks Total</td>
<td>8192</td>
<td></td>
</tr>
<tr>
<td>Blocks in Current Version</td>
<td>4954</td>
<td></td>
</tr>
<tr>
<td>Blocks in All Versions</td>
<td>4954</td>
<td></td>
</tr>
<tr>
<td>Blocks in Use</td>
<td>4986</td>
<td></td>
</tr>
<tr>
<td>% Blocks in Use</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>** Blocks Leaked**</td>
<td>32</td>
<td>*****</td>
</tr>
<tr>
<td>** Allocation Statistics**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blocks Created in Current TXN</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>Blocks To Drop in Current TXN</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>Marked Logical Blocks</td>
<td>8064</td>
<td></td>
</tr>
<tr>
<td>Marked Physical Blocks</td>
<td>4954</td>
<td></td>
</tr>
<tr>
<td>Marked Pages</td>
<td>504</td>
<td></td>
</tr>
<tr>
<td>Blocks in Freelist</td>
<td>126553</td>
<td></td>
</tr>
<tr>
<td>Imaginary Blocks</td>
<td>122567</td>
<td></td>
</tr>
<tr>
<td>Highest PBN in Use</td>
<td>5432</td>
<td></td>
</tr>
<tr>
<td>** 1st Unowned PBN**</td>
<td>452</td>
<td>*****</td>
</tr>
<tr>
<td>Total Free Blocks</td>
<td>4206</td>
<td></td>
</tr>
<tr>
<td>Usable Free Blocks</td>
<td>3125</td>
<td></td>
</tr>
<tr>
<td>% Free Space Fragmented</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Max Blocks Per Page</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>1 Block Page Count</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>3 Block Page Count</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>4 Block Page Count</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Block Hole Count</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>16 Block Hole Count</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>Database Objects Checked</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B-Array Count</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Blockmap Identity Count</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>** Connection Statistics**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>See also</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For more information on the use of sp_iqcheckdb and the interpretation of the sp_iqcheckdb output, see Chapter 13, “System Recovery and Database Repair,” in the *System Administration Guide: Volume I.*
sp_iqdroplogin procedure

Function: Drops a Sybase IQ user account.

Syntax1: `call sp_iqdroplogin ('userid')`
Syntax2: `sp_iqdroplogin 'userid'`
Syntax3: `sp_iqdroplogin userid`
Syntax4: `sp_iqdroplogin ('userid')`

Permissions: DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage: `userid` User ID of the user to drop.

Description: `sp_iqdroplogin` drops the specified user.

Examples: The following stored procedure calls remove the user `rose`.

```
sp_iqdroplogin 'rose'
sp_iqdroplogin rose
call sp_iqdroplogin ('rose')
```

See also: “sp_iqaddlogin procedure” on page 355

REVOKE statement in Reference: Statements and Options

Chapter 8, “Managing User IDs and Permissions,” in System Administration Guide: Volume 1

sp_iqemptyfile procedure

Function: Empties a dbspace file and moves the objects in the file to another available read-write dbspace file.

Syntax: `sp_iqemptyfile (logical-file--name)`

Permissions: DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.
**System stored procedures**

**Description**
sp_iqemptyfile empties a dbspace file. The dbspace must be read-only before you can execute the sp_iqemptyfile procedure. The procedure moves the objects in the file to another available read-write dbspace file. If there is not other read-write dbspace file available, then Sybase IQ displays an error message.

**Note** In a multiplex environment, you can run sp_iqemptyfile only on the coordinator.

There must be one read-write dbspace available in order for the procedure to succeed.

**Example**
The following empties dbspace dbspace1:

```sql
sp_iqemptyfile 'dbspace1'
```

### sp_iqestjoin procedure

**Function**
Estimates the space needed to create join indexes for the tables you specify.

**Syntax**
```
sp_iqestjoin ( table1_name, table1_row_#, table2_name, table2_row_#, relation, iq_page_size )
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Returns the amount of space a join index uses based on the tables being joined. This procedure assumes that the database was created with the default block size for the specified IQ page size (or else the estimate is incorrect).

If you specify unqualified table names, then ensure that you are the owner of the tables being joined. If you are not the table owner, then provide a qualified table name for each table, such as 'owner.tablename'.

Table 7-20 lists the sp_iqestjoin parameters.
Table 7-20: sp_iqestjoin parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table1_name</td>
<td>char(256)</td>
<td>Name of the first table in the join.</td>
</tr>
<tr>
<td>table1_row_#</td>
<td>int</td>
<td>Number of rows in the first table that participates in the join.</td>
</tr>
<tr>
<td>table2_name</td>
<td>char(256)</td>
<td>Name of the second table in the join.</td>
</tr>
<tr>
<td>table2_row_#</td>
<td>int</td>
<td>Number of rows in the second table that participates in the join.</td>
</tr>
<tr>
<td>relation</td>
<td>char(9)</td>
<td>Type of join, which can be “one&gt;&gt;many” or “one&gt;&gt;one” (do not leave any spaces between the words and the operator). The default is “one&gt;&gt;many”.</td>
</tr>
<tr>
<td>iq_page_size</td>
<td>smallint</td>
<td>The page size defined for the IQ segment of the database (must be a power of 2 between 1024 and 524288; the default is 131072).</td>
</tr>
</tbody>
</table>

Example

```sql
call sp_iqestjoin ( 'Customers', 1500000, 'SalesOrders', 15000000, 'one>>many', 65536 )
```

Cases

<table>
<thead>
<tr>
<th>Indexsize</th>
<th>Create time</th>
<th>Msg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table1: Customers
Rows: 1500000
Columns:
8
Width:
223

Table2: SalesOrders
Rows: 15000000
Columns:
9
Width:
134
IQpagesize:
65536
Min Case 48001024 3h0m/CPU
Max Case 95449088 9h6m/CPU
Avg Case 70496256 5h53m/CPU
**sp_iqestdbspaces procedure**

**Function**
Estimates the number and size of dbspaces needed for a given total index size.

**Syntax**
```
sp_iqestdbspaces ( db_size_in_bytes, iq_page_size,
                 min_#_of_bytes, max_#_of_bytes )
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Displays information about the number and size of dbspace segments based on the size of the database, the IQ page size, and the range of bytes per dbspace segment. This procedure assumes that the database was created with the default block size for the specified IQ page size (or else the estimate is incorrect). Table 7-21 lists the sp_iqestdbspaces parameters.

**Table 7-21: sp_iqestdbspaces parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db_size_in_bytes</td>
<td>decimal(16)</td>
<td>Size of the database in bytes.</td>
</tr>
<tr>
<td>iq_page_size</td>
<td>smallint</td>
<td>The page size defined for the IQ segment of the database (must be a power of 2 between 65536 and 524288; the default is 131072).</td>
</tr>
<tr>
<td>min_#_of_bytes</td>
<td>int</td>
<td>The minimum number of bytes per dbspace segment. The default is 20,000,000 (20MB).</td>
</tr>
<tr>
<td>max_#_of_bytes</td>
<td>int</td>
<td>The maximum number of bytes per dbspace segment. The default is 2,146,304,000 (2.146GB).</td>
</tr>
</tbody>
</table>

**Usage**
sp_iqestdbspaces displays four types of recommendations, depending on how much of the data is unique:

**min** If there is little variation in data, you can choose to create only the dbspace segments of the sizes recommended as min. These recommendations reflect the best possible compression on data with the least possible variation.

**avg** If your data has an average amount of variation, create the dbspace segments recommended as min, plus additional segments of the sizes recommended as avg.

**max** If your data has a high degree of variation (many unique values), create the dbspace segments recommended as min, avg, and max.

**spare** If you are uncertain about the number of unique values in your data, create the dbspace segments recommended as min, avg, max, and spare. You can always delete unused segments after loading your data, but creating too few can cost you some time.
Using `sp_iqestdbspaces` with other system stored procedures

1. Run `sp_iqestjoin` for all the table pairs you expect to join frequently.
2. Select one of the suggested index sizes for each pair of tables.
3. Total the index sizes you selected for all tables.
4. Run `sp_iqestspace` for all tables.
5. Total all of the RAW DATA index sizes returned by `sp_iqestspace`.
6. Add the total from step 3 to the total from step 5 to determine total index size.
7. Use the total index size calculated in step 6 as the `db_size_in_bytes` parameter in `sp_iqestdbspaces`.

Results of `sp_iqestdbspaces` are only estimates, based on the average size of an index. The actual size depends on the data stored in the tables, particularly on how much variation there is in the data.

Sybase strongly recommends that you create the spare dbspace segments, because you can delete them later if they are unused.

**Example**

```
sp_iqestdbspaces 12000000000, 65536, 500000000, 2146304000
```

<table>
<thead>
<tr>
<th>dbspace files</th>
<th>Type</th>
<th>Size</th>
<th>Msg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>min</td>
<td>2146304000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>min</td>
<td>2146304000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>min</td>
<td>507392000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>avg</td>
<td>2146304000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>max</td>
<td>2053697536</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>spare</td>
<td>1200001024</td>
<td></td>
</tr>
</tbody>
</table>

This example estimates the size and number of dbspace segments needed for a 12GB database. Sybase IQ recommends that you create a minimum of 3 segments (listed as min) for the best compression, if you expect little uniqueness in the data. If the data has an average amount of variation, 1 more segment (listed as avg) should be created. Data with a lot of variation (many unique values, requiring extensive indexing), may require 1 more segment (listed as max). You can ensure that your initial load succeeds by creating a spare segment of 1200001024 bytes. Once you have loaded the database, you can delete any unused dbspace segments.
System stored procedures

**sp_iqestspace procedure**

**Function**
Estimates the amount of space needed to create an index based on the number of rows in the table.

**Syntax**

```
sp_iqestspace ( table_name, #_of_rows, iq_page_size )
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Displays the amount of space that a database requires based on the number of rows in the underlying database tables and on the database IQ page size. This procedure assumes that the database was created with the default block size for the specified IQ page size (or else the estimate is incorrect). Table 7-22 lists the `sp_iqestspace` parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>table_name</code></td>
<td>char(256)</td>
<td>Name of the table</td>
</tr>
<tr>
<td><code>#_of_rows</code></td>
<td>int</td>
<td>Number of rows in the table</td>
</tr>
<tr>
<td><code>iq_page_size</code></td>
<td>smallint</td>
<td>The page size defined for the IQ segment of the database (must be a power of 2 between 65536 and 524288; the default is 131072)</td>
</tr>
</tbody>
</table>

**sp_iqevent procedure**

**Function**
Displays information about system and user-defined events.

**Syntax**

```
sp_iqevent [ event-name ], [ event-owner ], [ event-type ]
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**

- `event-name` The name of the event.
- `event-owner` The owner of the event.
- `event-type` The type of event. Allowed values are:
  - SYSTEM: displays information about system events (events owned by user SYS or dbo) only
  - ALL: displays information about user and system events
  - Any other value: displays information about user events
The sp_iqevent procedure can be invoked without any parameters. If no parameters are specified, only information about user events (events not owned by dbo or SYS) is displayed by default.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqevent NULL, NULL, SYSTEM` and `sp_iqevent NULL, user1`.

**Table 7-23: sp_iqevent usage examples**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sp_iqevent</code></td>
<td>Displays information about all user events in the database</td>
</tr>
<tr>
<td><code>sp_iqevent e1</code></td>
<td>Displays information about the event e1</td>
</tr>
<tr>
<td><code>sp_iqevent non_existing_event</code></td>
<td>No rows returned, as the event non_existing_event does not exist</td>
</tr>
<tr>
<td><code>sp_iqevent NULL, DBA</code></td>
<td>Displays information about all events owned by DBA</td>
</tr>
<tr>
<td><code>sp_iqevent e1, DBA</code></td>
<td>Displays information about the event e1 owned by DBA</td>
</tr>
<tr>
<td><code>sp_iqevent ev_iqbegintxn</code></td>
<td><code>ev_iqbegintxn</code> is a system-defined event. If there is no user-defined event also named <code>ev_iqbegintxn</code>, no rows are returned. (By default only user-defined events are returned.)</td>
</tr>
<tr>
<td><code>sp_iqevent ev_iqbegintxn, dbo</code></td>
<td>No rows returned, as the event <code>ev_iqbegintxn</code> is not a user event (by default only user events returned)</td>
</tr>
<tr>
<td><code>sp_iqevent NULL, NULL, SYSTEM</code></td>
<td>Displays information about all system events (owned by dbo or SYS)</td>
</tr>
<tr>
<td><code>sp_iqevent ev_iqbegintxn, NULL, SYSTEM</code></td>
<td>Displays information about the system event <code>ev_iqbegintxn</code></td>
</tr>
<tr>
<td><code>sp_iqevent ev_iqbegintxn, dbo, ALL</code></td>
<td>Displays information about the system event <code>ev_iqbegintxn</code> owned by dbo</td>
</tr>
</tbody>
</table>

**Description**

The sp_iqevent stored event displays information about events in a database. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if `event-name` is specified, only information about the specified event is displayed. If `event-owner` is specified, sp_iqevent only returns information about events owned by the specified owner. If no parameters are specified, sp_iqevent displays information about all the user events in the database.

The sp_iqevent procedure returns information in the following columns:
### Table 7-24: sp_iqevent columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_name</td>
<td>The name of the event</td>
</tr>
<tr>
<td>event_owner</td>
<td>The owner of the event</td>
</tr>
<tr>
<td>event_type</td>
<td>For system events, the event type as listed in the SYSEVENTTYPE system table</td>
</tr>
<tr>
<td>enabled</td>
<td>Indicates whether or not the event is allowed to fire (Y/N)</td>
</tr>
<tr>
<td>action</td>
<td>The event handler definition</td>
</tr>
<tr>
<td>condition</td>
<td>The WHERE condition used to control firing of the event handler</td>
</tr>
<tr>
<td>location</td>
<td>The location where the event is allowed to fire:</td>
</tr>
<tr>
<td>remarks</td>
<td>A comment string</td>
</tr>
</tbody>
</table>

### Examples

Display information about the user-defined event e1:

```sql
sp_iqevent e1
```

<table>
<thead>
<tr>
<th>event_name</th>
<th>event_owner</th>
<th>event_type</th>
<th>enabled</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>DBA</td>
<td>(NULL)</td>
<td>Y</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>condition</th>
<th>location</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>(NULL)</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

Display information about all system events:

```sql
sp_iqevent NULL, NULL, SYSTEM
```

<table>
<thead>
<tr>
<th>event_name</th>
<th>event_owner</th>
<th>event_type</th>
<th>enabled</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev_iqbegintxn</td>
<td>dbo</td>
<td>IQTLVAvailable</td>
<td>Y</td>
<td>begin call dbo.sp_iqlog...</td>
</tr>
<tr>
<td>ev_iqmpxcompact</td>
<td>dbo</td>
<td>(NULL)</td>
<td>N</td>
<td>begin Declare _Catalog...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>condition</th>
<th>location</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>(NULL)</td>
<td>A</td>
<td>(NULL)</td>
</tr>
<tr>
<td>(NULL)</td>
<td>A</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

### See also

CREATE EVENT statement in Reference: Statements and Options

Chapter 6, “Automating Tasks Using Schedules and Events” in the System Administration Guide: Volume 2
sp_iqfile procedure

**Function**
Displays detailed information about each dbfile in a dbspace.

**Syntax**
```
sp_iqfile [ dbspace-name ]
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
sp_iqfile displays the usage, properties, and types of data in each dbfile in a dbspace. You can use this information to determine whether data must be moved, and for data that has been moved, whether the old versions have been deallocated.

sp_iqfile displays the following information:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBSpaceName</td>
<td>Name of the dbspace as specified in the CREATE DBSPACE statement. Dbspace names are case-insensitive for databases created with CASE RESPECT.</td>
</tr>
<tr>
<td>DBFileName</td>
<td>Logical file name.</td>
</tr>
<tr>
<td>Path</td>
<td>Location of the physical file or raw partition.</td>
</tr>
<tr>
<td>SegmentType</td>
<td>Type of dbspace (MAIN or TEMPORARY).</td>
</tr>
<tr>
<td>RWMode</td>
<td>Mode of the dbspace: read-write (RW) or read-only (RO).</td>
</tr>
<tr>
<td>Online</td>
<td>T (online) or F (offline).</td>
</tr>
<tr>
<td>Usage</td>
<td>Percent of dbspace currently in use by this file in the dbspace.</td>
</tr>
<tr>
<td>DBFileSize</td>
<td>Current size of the file or raw partition. For a raw partition, this size value can be less than the physical size.</td>
</tr>
<tr>
<td>Reserve</td>
<td>Reserved space that can be added to this file in the dbspace.</td>
</tr>
<tr>
<td>StripeSize</td>
<td>Amount of data written to the file before moving to the next file, if disk striping is on.</td>
</tr>
<tr>
<td>BlkTypes</td>
<td>Space used by both user data and internal system structures (see Table 7-26 for identifier values).</td>
</tr>
<tr>
<td>FirstBlk</td>
<td>First IQ block number assigned to the file.</td>
</tr>
<tr>
<td>LastBlk</td>
<td>Last IQ block number assigned to the file.</td>
</tr>
<tr>
<td>OkToDrop</td>
<td>‘Y’ indicates the file can be dropped; otherwise ‘N’.</td>
</tr>
</tbody>
</table>

Table 7-26 lists the values of the block type identifiers.
System stored procedures

Table 7-26: sp_iqfile block types

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Block Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Active Version</td>
</tr>
<tr>
<td>B</td>
<td>Backup Structures</td>
</tr>
<tr>
<td>C</td>
<td>Checkpoint Log</td>
</tr>
<tr>
<td>D</td>
<td>Database Identity</td>
</tr>
<tr>
<td>F</td>
<td>Free list</td>
</tr>
<tr>
<td>G</td>
<td>Global Free list Manager</td>
</tr>
<tr>
<td>H</td>
<td>Header Blocks of the Free List</td>
</tr>
<tr>
<td>I</td>
<td>Index Advice Storage</td>
</tr>
<tr>
<td>M</td>
<td>Multiplex CM*</td>
</tr>
<tr>
<td>O</td>
<td>Old Version</td>
</tr>
<tr>
<td>T</td>
<td>Table Use</td>
</tr>
<tr>
<td>U</td>
<td>Index Use</td>
</tr>
<tr>
<td>N</td>
<td>Column Use</td>
</tr>
<tr>
<td>X</td>
<td>Drop at Checkpoint</td>
</tr>
</tbody>
</table>

*The multiplex commit identity block (actually 128 blocks) exists in all IQ databases, even though it is not used by simplex databases.

Example

Displays information about the files in the dbspaces:

```
sp_iqfile;
```

<table>
<thead>
<tr>
<th>DBSpace Name</th>
<th>DBFile Name</th>
<th>Path</th>
<th>Segment Type</th>
<th>RWMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQ_SYSTEM_MAIN</td>
<td>IQ_SYSTEM_MAIN</td>
<td>/sunopt/users/user1/iqdemo.iq</td>
<td>MAIN</td>
<td>RW</td>
</tr>
<tr>
<td>IQ_SYSTEM_TEMP</td>
<td>IQ_SYSTEM_TEMP</td>
<td>/sunopt/users/user1/iqdemo.iqtmp</td>
<td>TEMPORARY</td>
<td>RW</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Online Usage</th>
<th>DBFileSize</th>
<th>Reserve</th>
<th>Stripesize</th>
<th>BlkTypes</th>
<th>FirstBlk</th>
<th>LastBlk</th>
<th>OkToDrop</th>
<th>BlkTypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>21</td>
<td>300M</td>
<td>50M</td>
<td>8K</td>
<td>1H, 7648F, 1H, 64F, 16A</td>
<td>32D, 128M</td>
<td>1</td>
<td>38400</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>100M</td>
<td>50M</td>
<td>8K</td>
<td>1H, 64F, 16A</td>
<td>12800</td>
<td>1</td>
<td>12800</td>
</tr>
</tbody>
</table>

See also

“sp_iqdbspaceinfo procedure” on page 391, and “sp_iqindexinfo procedure” on page 421

Chapter 5, “Working with Database Objects,” in the Sybase IQ System Administration Guide

sp_iqhelp procedure

Function

Displays information about system and user-defined objects and data types.
CHAPTER 7  System Procedures

Syntax

`sp_iqhelp [ obj-name ], [ obj-owner ], [ obj-category ], [ obj-type ]`

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

- **obj-name**  The name of the object.
- **obj-owner**  The owner of the object.
- **obj-category**  An optional parameter that specifies the category of the object.

<table>
<thead>
<tr>
<th>object-type parameter</th>
<th>Specifies</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;table&quot;</td>
<td>The object is a base table</td>
</tr>
<tr>
<td>&quot;view&quot;</td>
<td>The object is an view</td>
</tr>
<tr>
<td>&quot;procedure&quot;</td>
<td>The object is a stored procedure or function</td>
</tr>
<tr>
<td>&quot;event&quot;</td>
<td>The object is an event</td>
</tr>
<tr>
<td>&quot;datatype&quot;</td>
<td>The object is a system or user-defined data type</td>
</tr>
</tbody>
</table>

Columns, constraints, and indexes are associated with tables and cannot be queried directly. When a table is queried, the information about columns, indexes, and constraints associated with that table is displayed.

If the specified object category is not one of the allowed values, an "Invalid object category" error is returned.

- **obj-type**  The type of object. Allowed values are:
  - SYSTEM: displays information about system objects (objects owned by user SYS or dbo) only
  - ALL: displays information about all objects

By default, only information about non-system objects is displayed. If the specified object type is not SYSTEM or ALL, an “Invalid object type” error is returned.

The `sp_iqhelp` procedure can be invoked without any parameters. If no parameters are specified, `sp_iqhelp` displays information about all independent objects in the database, that is, base tables, views, stored procedures, functions, events, and data types.

If you do not specify any of the first three parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqhelp NULL, NULL, NULL, SYSTEM` and `sp_iqhelp NULL, user1, "table"`.

Enclose the **obj-category** parameter in single or double quotes., except when NULL.
If `sp_iqhelp` does not find an object in the database that satisfies the specified description, the error "No object found for the given description" is returned.

### Table 7-28: sp_iqhelp usage examples

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sp_iqhelp</code></td>
<td>Displays summary information about all user-defined tables, views, procedures, events, and data types in the database</td>
</tr>
<tr>
<td><code>sp_iqhelp t1, u1, “table”</code></td>
<td>Displays information about table <code>t1</code> owned by user <code>u1</code> and the columns, indexes, and constraints associated with <code>t1</code></td>
</tr>
<tr>
<td><code>sp_iqhelp NULL, u1, “view”</code></td>
<td>Displays information about view <code>v1</code> owned by user <code>u1</code> and the columns associated with <code>v1</code></td>
</tr>
<tr>
<td><code>sp_iqhelp sp2</code></td>
<td>Displays information about the procedure <code>sp2</code> and the parameters of <code>sp2</code></td>
</tr>
<tr>
<td><code>sp_iqhelp e1</code></td>
<td>Displays information about the event <code>e1</code></td>
</tr>
<tr>
<td><code>sp_iqhelp dt1</code></td>
<td>Displays information about the data type <code>dt1</code></td>
</tr>
<tr>
<td><code>sp_iqhelp NULL, NULL, NULL, SYSTEM</code></td>
<td>Displays summary information about all system objects (owned by <code>dbo</code> or <code>SYS</code>)</td>
</tr>
<tr>
<td><code>sp_iqhelp non_existing_obj</code></td>
<td>Error &quot;Object 'non_existing_obj' not found&quot; returned, as the object <code>non_existing_obj</code> does not exist</td>
</tr>
<tr>
<td><code>sp_iqhelp NULL, non_existing_user</code></td>
<td>Error &quot;User 'non_existing_user' not found&quot; returned, as the user <code>non_existing_user</code> does not exist</td>
</tr>
<tr>
<td><code>sp_iqhelp t1, NULL, “apple”</code></td>
<td>Error &quot;Invalid object category 'apple'&quot; returned, as &quot;apple&quot; is not an allowed value for <code>obj-category</code></td>
</tr>
<tr>
<td><code>sp_iqhelp t1, NULL, NULL, “USER”</code></td>
<td>Error &quot;Invalid object type ‘USER’&quot; returned, as &quot;USER&quot; is not an allowed value for <code>obj-type</code></td>
</tr>
</tbody>
</table>

### Description

The `sp_iqhelp` stored procedure displays information about system and user-defined objects and data types in an IQ database. Objects supported by `sp_iqhelp` are tables, views, columns, indexes, join indexes, constraints, stored procedures, functions, events, and data types.
If you specify one or more parameters, the result is filtered by the specified parameters. For example, if `obj-name` is specified, only information about the specified object is displayed. If `obj-owner` is specified, `sp_iqhelp` returns information only about objects owned by the specified owner. If no parameters are specified, `sp_iqhelp` displays summary information about all user-defined tables, views, procedures, events, and data types in the database.

The `sp_iqhelp` procedure returns either summary or detailed information, depending on whether the specified parameters match multiple objects or a single object. The output columns of `sp_iqhelp` are similar to the columns displayed by the stored procedures `sp_iqtable`, `sp_iqindex`, `sp_iqview`, and `sp_iqconstraint`.

When multiple objects match the specified `sp_iqhelp` parameters, `sp_iqhelp` displays summary information about those objects.

<table>
<thead>
<tr>
<th>Object type</th>
<th>Columns displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>base table</td>
<td><code>table_name, table_owner, server_type, location,</code></td>
</tr>
<tr>
<td></td>
<td><code>table_constraints, remarks</code></td>
</tr>
<tr>
<td>view</td>
<td><code>view_name, view_creator, view_def, server_type,</code></td>
</tr>
<tr>
<td></td>
<td><code>location, remarks</code></td>
</tr>
<tr>
<td>stored procedure</td>
<td><code>proc_name, proc_creator, proc_defn, replicate,</code></td>
</tr>
<tr>
<td></td>
<td><code>srvid, remarks</code></td>
</tr>
<tr>
<td>function</td>
<td><code>proc_name, proc_creator, proc_defn, replicate,</code></td>
</tr>
<tr>
<td></td>
<td><code>remarks</code></td>
</tr>
<tr>
<td>event</td>
<td><code>event_name, event_creator, enabled, location,</code></td>
</tr>
<tr>
<td></td>
<td><code>event_type, action, external_action, condition,</code></td>
</tr>
<tr>
<td></td>
<td><code>remarks</code></td>
</tr>
<tr>
<td>system and user-defined data</td>
<td><code>type_name, creator, nulls, width, scale, default,</code></td>
</tr>
<tr>
<td>types</td>
<td><code>check</code></td>
</tr>
</tbody>
</table>

When a single object matches the specified `sp_iqhelp` parameters, `sp_iqhelp` displays detailed information about the object.
Table 7-30: sp_iqhelp detailed information

<table>
<thead>
<tr>
<th>Object type</th>
<th>Description</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Displays information about the specified base table, its columns, indexes, constraints, and join indexes (if the table participates in any join indexes)</td>
<td>• Table columns: table_name, table_owner, server_type, location, table_constraints, remarks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Column columns: column_name, domain_name, width, scale, nulls, default, check, pkey, user_type, cardinality, est_cardinality, remarks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Index columns: index_name, column_name, index_type, unique_index, location, remarks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Constraint columns: constraint_name (role), column_name, index_name, constraint_type, foreigntable_name, foreigntable_owner, foreigncolumn_name, foreignindex_name, location</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Join index columns: joinindex_name, creator, left_table_name, left_table_owner, left_column_name, join_type, right_table_name, right_table_owner, right_column_name, key_type, valid, remarks</td>
</tr>
<tr>
<td>view</td>
<td>Displays information about the specified view and its columns</td>
<td>• View columns: view_name, view_creator, view_def, server_type, location, remarks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Column columns: column_name, domain_name, width, scale, nulls, default, check, pkey, user_type, cardinality, est_cardinality, remarks</td>
</tr>
<tr>
<td>stored procedure</td>
<td>Displays information about the specified procedure and its parameters</td>
<td>• Procedure columns: proc_name, proc_creator, proc_defn, replicate, srvid, remarks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Parameter columns: parameter_name, type, width, scale, default, mode</td>
</tr>
<tr>
<td>function</td>
<td>Displays information about the specified function and its parameters</td>
<td>• Function columns: proc_name, proc_creator, proc_defn, replicate, srvid, remarks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Parameter columns: parameter_name, type, width, scale, default, mode</td>
</tr>
<tr>
<td>event</td>
<td>Displays information about the specified event</td>
<td>• Event columns: event_name, event_creator, enabled, location, event_type, action, external_action, condition, remarks</td>
</tr>
<tr>
<td>data type</td>
<td>Displays information about the specified data type</td>
<td>• Data type columns: type_name, creator, nulls, width, scale, default, check</td>
</tr>
</tbody>
</table>

For descriptions of the individual output columns listed in Table 7-30, refer to the descriptions of the following stored procedures:

- table: “sp_iqtable procedure” on page 472
Adaptive Server Enterprise compatibility  The Sybase IQ sp_iqhelp stored procedure is similar to the Adaptive Server Enterprise sp_help procedure, which displays information about any database object listed in the SYSOBJECTS system table and about system and user-defined data types.

Sybase IQ has some architectural differences from Adaptive Server in terms of types of objects supported and the namespace of objects. In Adaptive Server, all objects (tables, views, stored procedures, logs, rules, defaults, triggers, check constraints, referential constraints, and temporary objects) are stored in the SYSOBJECTS system table and are in the same namespace. The objects supported by Sybase IQ (tables, views, stored procedures, events, primary keys, and unique, check, and referential constraints) are stored in different system tables and are in different namespaces. For example, in Sybase IQ a table can have the same name as an event or a stored procedure.

Because of the architectural differences between Sybase IQ and Adaptive Server, the types of objects supported by and the syntax of Sybase IQ sp_iqhelp are different from the supported objects and syntax of Adaptive Server sp_help; however, the type of information about database objects that is displayed by both stored procedures is similar.
Examples

Display detailed information about the table `sale`:

```
sp_iqhelp sale
```

<table>
<thead>
<tr>
<th>Table_name</th>
<th>Table_owner</th>
<th>Server_type</th>
<th>Location</th>
<th>dbspace_id</th>
<th>isPartitioned</th>
</tr>
</thead>
<tbody>
<tr>
<td>sale</td>
<td>DBA</td>
<td>IQ</td>
<td>Main</td>
<td>16387</td>
<td>N</td>
</tr>
</tbody>
</table>

Remarks on `table_constraints`

<table>
<thead>
<tr>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>column_name</th>
<th>domain_name</th>
<th>width</th>
<th>scale</th>
<th>nulls</th>
<th>default</th>
<th>cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod_id</td>
<td>integer</td>
<td>4</td>
<td>0</td>
<td>Y</td>
<td>(NULL)</td>
<td>0</td>
</tr>
<tr>
<td>month_num</td>
<td>integer</td>
<td>4</td>
<td>0</td>
<td>Y</td>
<td>(NULL)</td>
<td>0</td>
</tr>
<tr>
<td>rep_id</td>
<td>integer</td>
<td>4</td>
<td>0</td>
<td>Y</td>
<td>(NULL)</td>
<td>0</td>
</tr>
<tr>
<td>sales</td>
<td>integer</td>
<td>4</td>
<td>0</td>
<td>Y</td>
<td>(NULL)</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>est_cardinality</th>
<th>isPartitioned</th>
<th>remarks</th>
<th>check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
<tr>
<td>0</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
<tr>
<td>0</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
<tr>
<td>0</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index_name</th>
<th>column_name</th>
<th>index_type</th>
<th>unique_index</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIQ_IDX_T463_C2_FP</td>
<td>month_num</td>
<td>FP</td>
<td>N</td>
<td>Main</td>
</tr>
<tr>
<td>ASIQ_IDX_T463_C1_FP</td>
<td>prod_id</td>
<td>FP</td>
<td>N</td>
<td>Main</td>
</tr>
<tr>
<td>ASIQ_IDX_T463_C3_FP</td>
<td>rep_id</td>
<td>FP</td>
<td>N</td>
<td>Main</td>
</tr>
<tr>
<td>ASIQ_IDX_T463_C4_FP</td>
<td>sales</td>
<td>FP</td>
<td>N</td>
<td>Main</td>
</tr>
</tbody>
</table>

Remarks on `index_name`

<table>
<thead>
<tr>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
</tr>
</tbody>
</table>

Display detailed information about the procedure `sp_customer_list`:

```
sp_iqhelp sp_customer_list
```
sp_customer_list   DBA    create procedure DBA.sp_customer_list()
result(id integer company_name char(35))
begin
select id company_name from Customers
end

parm_name         parm_type parm_mode domain_name width scale
=======           ========= ========= ===========  ===== =====
id              result out integer        4  0
company_name result out char           35  0

default
========
(Number)

sp_iqindex and sp_iqindex_alt procedures
Function     Lists information about indexes.
Syntax 1     \texttt{sp_iqindex ( [ table\_name ],[column\_name ],[table\_owner ] )}
Syntax 2     \texttt{sp_iqindex [table\_name=’tablename’],
                [column\_name=’columnname’],[table\_owner=’tableowner’]}
Syntax 3     \texttt{sp_iqindex\_alt ( [ table\_name ],[column\_name ],[table\_owner ] )}
Syntax 4     \texttt{sp_iqindex\_alt [table\_name=’tablename’],
                [column\_name=’columnname’],[table\_owner=’tableowner’]}
Permissions DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.
Usage
Syntax1     If you do not specify either of the first two parameters, but specify
the next parameter in the sequence, you must substitute NULL for the omitted
parameters. For example, \texttt{sp_iqindex NULL, NULL, DBA and sp_iqindex}
Departments, NULL, DBA.
Syntax2     You can specify the parameters in any order. Enclose them in single
quotes.
Syntax 3 and 4 Produces slightly different output when a multicol
umn index is present. Allows the same options as Syntax 1 and 2.
Displays information about indexes in the database. Specifying one of the parameters returns the indexes from only that table, column, or tables owned by the specified user. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all indexes for all tables in the database.

**Table 7-31: sp_iqindex and sp_iqindex_alt columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table</td>
</tr>
<tr>
<td>table_owner</td>
<td>The owner of the table</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column; multiple names can appear in a multicolumn index</td>
</tr>
<tr>
<td>index_type</td>
<td>The abbreviated index type (for example, HG, LF)</td>
</tr>
<tr>
<td>index_name</td>
<td>The name of the index</td>
</tr>
<tr>
<td>unique_index</td>
<td>'U' indicates the index is a unique index; otherwise, 'N'</td>
</tr>
<tr>
<td>location</td>
<td>TEMP = IQ temporary store, MAIN = IQ store, SYSTEM = catalog store</td>
</tr>
<tr>
<td>remarks</td>
<td>User comments added with the COMMENT statement</td>
</tr>
</tbody>
</table>

sp_iqindex always produces one line per index. sp_iqindex_alt produces one line per index per column if there is a multicolumn index.

**Examples**

The following variations in syntax both return all indexes on columns with the name DepartmentID:

```sql
call sp_iqindex (NULL,'DepartmentID')
sp_iqindex column_name='DepartmentID'
```

<table>
<thead>
<tr>
<th>table_name</th>
<th>column_name</th>
<th>index_type</th>
<th>index_name</th>
<th>unique_index</th>
<th>location</th>
<th>dbspace_id</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Departments</td>
<td>DepartmentID</td>
<td>FP</td>
<td>ASIQ_IDX_T201_C1_FP</td>
<td>N</td>
<td>Main</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
<tr>
<td>Departments</td>
<td>DepartmentID</td>
<td>HG</td>
<td>ASIQ_IDX_T201_C1_HG</td>
<td>U</td>
<td>Main</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
<tr>
<td>Employees</td>
<td>DepartmentID</td>
<td>FP</td>
<td>ASIQ_IDX_T202_C5_FP</td>
<td>N</td>
<td>Main</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

The following variations in syntax both return all indexes in the table Departments that is owned by table owner GROUPO:

```sql
sp_iqindex Departments,NULL,GROUPO
sp_iqindex table_name='Departments',table_owner='DBA'
```
The following variations in syntax for `sp_iqindex_alt` both return indexes on the table `Employees` that contain the column `City`. The index `emp_loc` is a multicolumn index on the columns `City` and `State`. `sp_iqindex_alt` displays one row per column for a multicolumn index.

```
sp_iqindex_alt Employees,City
sp_iqindex_alt table_name='Employees',
column_name='City'
```

<table>
<thead>
<tr>
<th>table_name</th>
<th>table_owner</th>
<th>column_name</th>
<th>index_type</th>
<th>index_name</th>
<th>unique_index</th>
<th>location</th>
<th>dbspaceno</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Departments</td>
<td>GRO UPO</td>
<td>Department</td>
<td>FP</td>
<td>ASIQ_IDX_T201_C3_</td>
<td>N</td>
<td>Main</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
<tr>
<td>Departments</td>
<td>GRO UPO</td>
<td>Department</td>
<td>FP</td>
<td>ASIQ_IDX_T201_C1_</td>
<td>N</td>
<td>Main</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
<tr>
<td>Departments</td>
<td>GRO UPO</td>
<td>Department</td>
<td>HG</td>
<td>ASIQ_IDX_T201_C1_</td>
<td>U</td>
<td>Main</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
<tr>
<td>Departments</td>
<td>GRO UPO</td>
<td>Department</td>
<td>FP</td>
<td>ASIQ_IDX_T201_C2_</td>
<td>N</td>
<td>Main</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

The output from `sp_iqindex` for the same table and column is slightly different:

```
sp_iqindex Employees,City
sp_iqindex Employees City
```

<table>
<thead>
<tr>
<th>table_name</th>
<th>table_owner</th>
<th>column_name</th>
<th>index_type</th>
<th>index_name</th>
<th>unique_index</th>
<th>dbspaceno</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employees</td>
<td>GRO UPO</td>
<td>City</td>
<td>FP</td>
<td>ASIQ_IDX_T452_C7_</td>
<td>N</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
<tr>
<td>Employees</td>
<td>GRO UPO</td>
<td>City</td>
<td>HG</td>
<td>emp_loc</td>
<td>N</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
<tr>
<td>Employees</td>
<td>GRO UPO</td>
<td>State</td>
<td>HG</td>
<td>emp_loc</td>
<td>N</td>
<td>16387</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

Reference: Building Blocks, Tables, and Procedures
System stored procedures

See also

“FP_LOOKUP_SIZE option,” “INDEX_ADVISOR option,” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in Reference: Statements and Options


sp_iqindexadvice procedure

Function Displays stored index advice messages. Optionally clears advice storage.

Syntax

sp_iqindexadvice ( [ resetflag ] )

Permissions DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage resetflag Lets the caller clear the index advice storage. If resetflag is nonzero, all advice is removed after the last row has been retrieved.

Description Allows users to query aggregated index advisor messages using SQL. Information can be used to help decide which indexes or schema changes will affect the most queries.

Table 7-32 lists INDEX_ADVISOR columns.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advice</td>
<td>Unique advice message</td>
</tr>
<tr>
<td>NInst</td>
<td>Number of instances of message</td>
</tr>
<tr>
<td>LastDT</td>
<td>Last date/time advice was generated</td>
</tr>
</tbody>
</table>

Examples Table 7-33 illustrates sample output from the sp_iqindexadvice procedure.
Table 7-33: Sample sp_iqindexadvice output

<table>
<thead>
<tr>
<th>Advice</th>
<th>NInst</th>
<th>LastDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a CMP index on DBA.tb (c2, c3) Predicate: (tb.c2 = tb.c3)</td>
<td>2073</td>
<td>2009-04-07 16:37:31.000</td>
</tr>
<tr>
<td>Convert HG index on DBA.tb.c4 to a unique HG</td>
<td>812</td>
<td>2009-04-06 10:01:15.000</td>
</tr>
<tr>
<td>Join Key Columns DBA.ta.c1 and DBA.tb.c1 have mismatched data types</td>
<td>911</td>
<td>2009-02-25 20:59:01.000</td>
</tr>
</tbody>
</table>

See also


“FP_LOOKUP_SIZE option,” “INDEX_ADVISOR option,” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in Reference: Statements and Options


sp_iqindexfragmentation procedure

Function

Reports information about the percentage of page space taken up within the B-trees, garrays, and bitmap structures in Sybase IQ indexes.

For garrays, the fill percentage calculation does not take into account the reserved space within the garray groups, which is controlled by the GARRAY_FILL_FACTOR_PERCENT option.

Syntax

dbo.sp_iqindexfragmentation ( 'target' )

target: table table-name | index index-name [...]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

table-name  Target table table-name reports on all nondefault indexes in the named table.

index-name  Target index index-name reports on the named index. Each index-name is a qualified index name. You can specify multiple indexes within the table, but you must repeat the index keyword with each index specified.
System stored procedures

Example

Reports the internal index fragmentation for nonunique HG index cidhg in table Customers:

    dbo.sp_iqindexfragmentation ('index customers.cirdhg')

<table>
<thead>
<tr>
<th>Index</th>
<th>Index type</th>
<th>btree node pages</th>
<th>GARRAY_FILL_FACTOR_PERCENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>dba.customers.cidhg</td>
<td>HG</td>
<td>3</td>
<td>75</td>
</tr>
<tr>
<td>SQLCODE</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fill Percent</th>
<th>btree pages</th>
<th>garray pages</th>
<th>bitmap pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 10%</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11 - 20%</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21 - 30%</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>31-40%</td>
<td>0</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>41 - 50%</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>51 - 60%</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>61 - 70%</td>
<td>2</td>
<td>0</td>
<td>120</td>
</tr>
<tr>
<td>71 - 80%</td>
<td>138</td>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>81 - 90%</td>
<td>24</td>
<td>122</td>
<td>14</td>
</tr>
<tr>
<td>91 - 100%</td>
<td>18</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

According to this output, of the 182 B-tree pages in nonunique HG index cidhg, 2 are between 61% and 70% full, 138 are 71% to 80% full, 24 are 81% - 90% full, and 18 are 91% - 100% full. Usage for garray and bitmap pages is reported in the same manner. All percentages are truncated to the nearest percentage point. HG indexes also display the value of option GARRAY_FILL_FACTOR_PERCENT. Those index types that use a B-tree also display the number of node (nonleaf) pages. These are HG, LF, WD, DATE, and DTTM.

If an error occurred during execution of the stored procedure for this index, the SQLCODE would be nonzero.

See also


“FP_LOOKUP_SIZE option,” “INDEX_ADVISOR option,” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in Reference: Statements and Options

**sp_iqindexinfo procedure**

**Function**
Displays the number of blocks used per index per main dbspace for a given object. If the object resides on several dbspaces, sp_iqindexinfo returns the space used in all dbspaces, as shown in the example.

**Syntax**
```
sp_iqindexinfo '{ [database ] [table table-name | index index-name ] [...] } [resources resource-percent ]'
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**
You can request index information for the entire database, or you can specify any number of table or index parameters. If a table name is specified, sp_iqindexinfo returns information on all indexes in the table. If an index name is specified, only the information on that index is returned.

You cannot specify a join index by name. Use the database keyword to display join indexes.

If the specified table-name or index-name is ambiguous or the object cannot be found, an error is returned.

By default in a multiplex database, sp_iqindexinfo displays information about the shared IQ store on a secondary node. If individual tables or indexes are specified, the store to display is automatically selected.

**Description**
sp_iqindexinfo shows the DBA on which dbspaces a given object resides. The DBA can use this information to determine which dbspaces must be given relocate mode to relocate the object.

The results of sp_iqindexinfo are from the point of view of the version seen by the transaction running the command. Blocks used by other versions are not shown.

**Table 7-34: sp_iqindexinfo columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Table, index, or join index name</td>
</tr>
<tr>
<td>Dbspace_name</td>
<td>Name of the dbspace</td>
</tr>
<tr>
<td>ObjSize</td>
<td>Size of data for this object on this dbspace</td>
</tr>
<tr>
<td>DBSpPct</td>
<td>Percent of dbspace used by this object</td>
</tr>
<tr>
<td>MinBlk</td>
<td>First block used by this object on this dbspace</td>
</tr>
</tbody>
</table>
System stored procedures

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxBlk</td>
<td>Last block used by this object on this dbspace; useful for determining which objects must be relocated before the dbspace is resized to a smaller size</td>
</tr>
</tbody>
</table>
Examples

Displays index information about the table t2:

```
sp_iqindexinfo 'table t2';
```

<table>
<thead>
<tr>
<th>Object</th>
<th>dbspace_name</th>
<th>ObjSize</th>
<th>DBSpPct</th>
<th>MinBlk</th>
<th>MaxBlk</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>IQ_SYSTEM_MAIN</td>
<td>32K</td>
<td>1</td>
<td>84</td>
<td>107</td>
</tr>
<tr>
<td>t2</td>
<td>dbspacedb2</td>
<td>160K</td>
<td>2</td>
<td>1045495</td>
<td>1045556</td>
</tr>
<tr>
<td>t2</td>
<td>dbspacedb3</td>
<td>8K</td>
<td>1</td>
<td>2090930</td>
<td>2090930</td>
</tr>
<tr>
<td>t2.DBA.ASIQ_IDX_T430_C1_FP</td>
<td>IQ_SYSTEM_MAIN</td>
<td>136K</td>
<td>2</td>
<td>126</td>
<td>321</td>
</tr>
<tr>
<td>t2.DBA.ASIQ_IDX_T430_C1_FP</td>
<td>dbspacedb3</td>
<td>152K</td>
<td>2</td>
<td>2091032</td>
<td>2091053</td>
</tr>
<tr>
<td>t2.DBA.t2c1hmng</td>
<td>dbspacedb2</td>
<td>136K</td>
<td>2</td>
<td>1045537</td>
<td>1045553</td>
</tr>
</tbody>
</table>

See also


Chapter 5, “Working with Database Objects” in the System Administration Guide: Volume 1


**sp_iqindexmetadata procedure**

**Function**
Displays index metadata for a given index. You can optionally restrict the output to only those indexes on a specified table, and to only those indexes belonging to a specified owner.

**Syntax**
```
dbo.sp_iqindexmetadata { 'index-name' [ , 'table-name' [ , 'owner-name' ] ] }
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**
Specifying a table name limits output to those indexes belonging to that table. Specifying an owner name limits output to indexes owned by that owner. Omitted parameters default to NULL. You can specify only one index per procedure.

**Description**
The first row of output is the owner name, table name, and index name for the index.

Subsequent rows of output are specific to the type of index specified.
### System stored procedures

**Table 7-35: sp_iqindexmetadata output rows**

<table>
<thead>
<tr>
<th>Index type</th>
<th>Metadata returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP, DATE, DTTM, TIME</td>
<td>Type, Version, LookupPages, Style, LookupEntries, 1stLookupPage, LargeLOBs, SmallLOBs, IQ Unique, LOB Compression (only if column datatype is LONG VARCHAR or LONG BINARY)</td>
</tr>
<tr>
<td>FP</td>
<td>Type, Version, Distinct Keys</td>
</tr>
<tr>
<td>HG</td>
<td>Type, Version, Distinct Keys</td>
</tr>
<tr>
<td>HNG</td>
<td>Type, Version, BitsPerBlockmap, NumberOfBits</td>
</tr>
<tr>
<td>LD</td>
<td>Type, Version&lt;ld&gt;, Version, Distinct Keys</td>
</tr>
<tr>
<td>LF</td>
<td>Type, Version, IndexStatus, NumberOfBlockmaps, BitsPerBlockmap, Distinct Keys</td>
</tr>
<tr>
<td>WD</td>
<td>Type, Version, KeySize, Delimiters, DelimiterCount, MaxKeyWordLength, PermitEmptyWord</td>
</tr>
</tbody>
</table>

**Examples**

The following command displays index information about the HG index `hg_index_col54`:

```sql
sp_iqindexmetadata 'hg_index_col54', 'metal', 'DBA';
```

```
'DBA', 'metal' 'hg_index_col54'
'Type', 'HG', ''
'Version', '2', ''
'Distinct Keys', '0', ''
```

**See also**

- “sp_iqindex and sp_iqindex_alt procedures” on page 415, “sp_iqindexfragmentation procedure” on page 419, “sp_iqindexinfo procedure” on page 421, and “sp_iqindexsize procedure” on page 425
- “FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in Reference: Statements and Options.
**sp_iqindexsize procedure**

**Function**
Gives the size of the specified index.

**Syntax**
```
sp_iqindexsize [[ owner.] table.] index_name
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>Index owner.</td>
</tr>
<tr>
<td>Indexname</td>
<td>Index for which results are returned, including the table name.</td>
</tr>
<tr>
<td>Type</td>
<td>Index type.</td>
</tr>
<tr>
<td>Info</td>
<td>Component of the IQ index for which the KBytes, Pages, and Compressed Pages are being reported. The components vary by index type. For example, the default (FP) index includes BARRAY (barray) and Bitmap (bm) components. The Low_Fast (LF) index includes B-tree (bt) and Bitmap (bm) components.</td>
</tr>
<tr>
<td>KBytes</td>
<td>Physical object size in KB.</td>
</tr>
<tr>
<td>Pages</td>
<td>Number of IQ pages needed to hold the object in memory.</td>
</tr>
<tr>
<td>Compressed Pages</td>
<td>Number of IQ pages when the object is compressed (on disk).</td>
</tr>
</tbody>
</table>

Returns the total size of the index in bytes and kilobytes, and an Info column that describes the component of the IQ index for which the KBytes, Pages, and Compressed Pages are reported. The components described vary by index type. For example, the default (FP) index includes BARRAY (barray) and Bitmap (bm) components. The Low_Fast (LF) index includes B-tree (bt) and Bitmap (bm) components.

Also returns the number of pages required to hold the object in memory and the number of IQ pages when the index is compressed (on disk).

You must specify the `index_name` parameter with this procedure. To restrict results to this index name in a single table, include `owner:table` when specifying the index.

**Example**
```
sp_iqindexsize ASIQ_IDX_T452_C19_FP
```

<table>
<thead>
<tr>
<th>Username</th>
<th>Indexname</th>
<th>Type</th>
<th>Info</th>
<th>KBytes</th>
<th>Pages</th>
<th>Compressed Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA</td>
<td>Employees.ASIQ_IDX_T452_C19_FP</td>
<td>FP</td>
<td>Total</td>
<td>288</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Reference: Building Blocks, Tables, and Procedures
### System stored procedures

#### sp_iqindexuse procedure

**Function**
Reports detailed usage information for secondary (non-FP) indexes accessed by the workload.

**Syntax**
```
sp_iqindexuse
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Each secondary index accessed by the workload displays a row. Indexes that have not been accessed do not appear. Index usage is broken down by optimizer, constraint, and query usage.

Indexes from tables created in SYSTEM are not reported.

---

<table>
<thead>
<tr>
<th>Username</th>
<th>Indexname</th>
<th>Type</th>
<th>Info</th>
<th>KBytes</th>
<th>Pages</th>
<th>Compressed Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Employees.ASIQ_IDX_T452_C19_FP</td>
<td>FP</td>
<td>vdo</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Employees.ASIQ_IDX_T452_C19_FP</td>
<td>FP</td>
<td>bt</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Employees.ASIQ_IDX_T452_C19_FP</td>
<td>FP</td>
<td>garray</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Employees.ASIQ_IDX_T452_C19_FP</td>
<td>FP</td>
<td>bm</td>
<td>136</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Employees.ASIQ_IDX_T452_C19_FP</td>
<td>FP</td>
<td>barray</td>
<td>152</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Employees.ASIQ_IDX_T452_C19_FP</td>
<td>FP</td>
<td>dpstore</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Employees.ASIQ_IDX_T452_C19_FP</td>
<td>FP</td>
<td>largelob</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

See also
Chapter 6, “Using Sybase IQ Indexes” in *System Administration Guide: Volume 1*.

“FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in *Reference: Statements and Options*. 

---

426 Sybase IQ
Table 7-37: sp_iqindexuse columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndexName</td>
<td>Index name</td>
</tr>
<tr>
<td>TableName</td>
<td>Table name</td>
</tr>
<tr>
<td>Owner</td>
<td>User name of index owner</td>
</tr>
<tr>
<td>UID**</td>
<td>Index unique identifier</td>
</tr>
<tr>
<td>Type</td>
<td>Index type</td>
</tr>
<tr>
<td>LastDT</td>
<td>Date/time of last access</td>
</tr>
<tr>
<td>NOpt</td>
<td>Number of metadata/uniqueness accesses</td>
</tr>
<tr>
<td>NQry</td>
<td>Number of query accesses</td>
</tr>
<tr>
<td>NConstraint</td>
<td>Number of accesses for unique or referential integrity checks</td>
</tr>
</tbody>
</table>

**UID is a number assigned by the system that uniquely identifies the instance of the index (where instance is defined when an object is created).**

Example

Sample output from the sp_iqindexuse procedure.

<table>
<thead>
<tr>
<th>IndexName</th>
<th>TableName</th>
<th>Owner</th>
<th>UID</th>
<th>Type</th>
<th>LastDT</th>
<th>NOpt</th>
<th>NQry</th>
<th>NConstraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_nationkey_hg</td>
<td>nation</td>
<td>DBA</td>
<td>29</td>
<td>HG</td>
<td>20070917 22:08:06~</td>
<td>12</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>n_regionkey_hg</td>
<td>nation</td>
<td>DBA</td>
<td>31</td>
<td>HG</td>
<td>20070917 22:08:06~</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r_regionkey_hg</td>
<td>region</td>
<td>DBA</td>
<td>47</td>
<td>HG</td>
<td>20070917 22:08:06~</td>
<td>12</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>s_suppkey_hg</td>
<td>supplier</td>
<td>DBA</td>
<td>64</td>
<td>HG</td>
<td>20070917 22:08:06~</td>
<td>12</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>p_partkey_hg</td>
<td>part</td>
<td>DBA</td>
<td>87</td>
<td>HG</td>
<td>20070917 22:08:06~</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>s_suppkey_hg</td>
<td>supplier</td>
<td>DBA</td>
<td>64</td>
<td>HG</td>
<td>20070917 22:08:06~</td>
<td>12</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also


“INDEX_ADVISOR option” in Chapter 2, “Database Options,” in *Reference: Statements and Options*

sp_iqjoinindex procedure

Function

Displays information about join indexes.

Syntax

```
sp_iqjoinindex [ left-table-name ], [ left-column-name ], [ left-table-owner ], [ right-table-name ], [ right-column-name ], [ right-table-owner ]
```
Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

left-table-name  The name of the table that forms the left side of the join operation.

left-column-name  The name of the column that is part of the left side of the join.

left-table-owner  The owner of the table that forms the left side of the join operation.

right-table-name  The name of the table that forms the right side of the join operation.

right-column-name  The name of the column that is part of the right side of the join.

right-table-owner  The owner of the table that forms the right side of the join operation.

The sp_iqjoinindex procedure can be invoked without any parameters. If no parameters are specified, sp_iqjoinindex displays information about all join indexes on IQ base tables. Note that join index tables are always IQ base tables. Join index tables cannot be temporary tables, remote tables, or proxy tables.

If you do not specify any of the first five parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, sp_iqjoinindex NULL, NULL, NULL, t2, n2, DB' and sp_iqjoinindex t1, NULL, NULL, t2.
Table 7-38: sp_iqjoinindex usage examples

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_iqjoinindex</td>
<td>Displays information about all the join indexes</td>
</tr>
<tr>
<td>sp_iqjoinindex t1, NULL, DBA</td>
<td>Displays information about all join indexes in which t1 owned by DBA forms the left side of the operation</td>
</tr>
<tr>
<td>sp_iqjoinindex t2, n1, DBA</td>
<td>Displays join index information with column n1 of table t2 owned by DBA as left side of the join</td>
</tr>
<tr>
<td>sp_iqjoinindex NULL, NULL, DBA, NULL, NULL, DBA</td>
<td>Displays information about all join indexes in which the left and right side tables are owned by DBA</td>
</tr>
<tr>
<td>sp_iqjoinindex NULL, NULL, n2, NULL, NULL</td>
<td>Displays information about all join indexes in which the table t2 is on the right side of the join operation</td>
</tr>
<tr>
<td>sp_iqjoinindex t1, n1, DBA, t2, n1, DBA</td>
<td>Displays information about join indexes in which the left side is column n1 of table t1 owned by DBA and the right side is column n1 of table t2 owned by DBA</td>
</tr>
<tr>
<td>sp_iqjoinindex non_existing_table</td>
<td>No rows returned, as the table non_existing_table does not exist</td>
</tr>
<tr>
<td>sp_iqjoinindex NULL, NULL, non_existing_user</td>
<td>No rows returned, as the user non_existing_user does not exist</td>
</tr>
</tbody>
</table>

See also

CREATE JOIN INDEX statement in Reference: Statements and Options

Chapter 6, “Using Sybase IQ Indexes” in the System Administration Guide: Volume 1

Description

The sp_iqjoinindex stored procedure displays information about join indexes in a database. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if left-table-name is specified, sp_iqjoinindex displays all the join indexes in which that table forms the left side of the join. If left-table-owner is specified, sp_iqjoinindex only returns join indexes in which the left table is owned by the specified owner. If no parameters are specified, sp_iqjoinindex displays information about all join indexes in the database.

The sp_iqjoinindex procedure returns information in the following columns:
System stored procedures

**Table 7-39: sp_iqjoinindex columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>joinindex_name</td>
<td>The name of the join index.</td>
</tr>
<tr>
<td>creator</td>
<td>The owner of the join index.</td>
</tr>
<tr>
<td>left_table_name</td>
<td>The name of the table that forms the left side of the join operation.</td>
</tr>
<tr>
<td>left_table_owner</td>
<td>The name of the owner of the table that forms the left side of the join operation.</td>
</tr>
<tr>
<td>left_column_name</td>
<td>The name of the column that is part of the left side of the join.</td>
</tr>
<tr>
<td>join_type</td>
<td>The only currently supported value is “=”</td>
</tr>
<tr>
<td>right_table_name</td>
<td>The name of the table that forms the right side of the join operation.</td>
</tr>
<tr>
<td>right_table_owner</td>
<td>The name of the owner of the table that forms the right side of the join operation.</td>
</tr>
<tr>
<td>right_column_name</td>
<td>The name of the column that is part of the right side of the join.</td>
</tr>
<tr>
<td>key_type</td>
<td>Defines the type of join on the keys:</td>
</tr>
<tr>
<td></td>
<td>• NATURAL: a natural join</td>
</tr>
<tr>
<td></td>
<td>• KEY: a key join</td>
</tr>
<tr>
<td></td>
<td>• ON: a left outer/right outer/full join</td>
</tr>
<tr>
<td>valid</td>
<td>Indicates whether this join index needs to be synchronized.</td>
</tr>
<tr>
<td></td>
<td>‘Y’ means that it does not require synchronization; ‘N’ means that it does require synchronization.</td>
</tr>
<tr>
<td>remarks</td>
<td>A comment string.</td>
</tr>
<tr>
<td>dbspace_id</td>
<td>Name of the dbspace in which specified join indexes reside.</td>
</tr>
</tbody>
</table>

**Examples**

Displays information about the join index in which table t1 forms the left side of the join operation:

```sql
sp_iqjoinindex t1
```

joinindex_name creator left_table_name left_table_owner left_column_name
join_type right_table_name right_table_owner right_column_name key_type
valid dbspace_id remarks
---
t1_t2_t3_join DBA t1 DBA n1
= t2 DBA n1 NATURAL
Y 16387 (NULL)

Displays information about the join index in which table t2 forms the left side of the join operation:

```sql
sp_iqjoinindex t2
```
Displays information about join indexes in which the left side is column name of table t2 owned by DBA and the right side is column name of table t3 owned by DBA:

```
sp_iqjoinindex t2, name, DBA, t3, name, DBA
```

### sp_iqjoinindexsize procedure

**Function**

Gives the size of the specified join index.

**Syntax**

```
sp_iqjoinindexsize (join_index_name)
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**

Returns the total size of the index in bytes, KBytes, and NBlocks (IQ blocks). Also returns the number of pages required to hold the join index in memory and the number of IQ pages when the join index is compressed (on disk). You must specify the `join_index_name` parameter with this procedure.
System stored procedures

Table 7-40: sp_iqjoinindexsize columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>Owner of the join index</td>
</tr>
<tr>
<td>JoinIndexName</td>
<td>Join index for which results are returned</td>
</tr>
<tr>
<td>Number of Tables</td>
<td>Number of tables in the join index</td>
</tr>
<tr>
<td>KBytes</td>
<td>Physical object size in KB</td>
</tr>
<tr>
<td>Pages</td>
<td>Number of IQ pages needed to hold the object in memory</td>
</tr>
<tr>
<td>Compressed Pages</td>
<td>Number of IQ pages when the object is compressed (on disk)</td>
</tr>
<tr>
<td>NBlocks</td>
<td>Number of IQ blocks</td>
</tr>
</tbody>
</table>

Example

```sql
sp_iqjoinindexsize ('t1t2')
```

<table>
<thead>
<tr>
<th>Username</th>
<th>JoinIndexName</th>
<th>Number of Tables</th>
<th>KBytes</th>
<th>Pages</th>
<th>Compressed Pages</th>
<th>NBlocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA</td>
<td>t1t2</td>
<td>2</td>
<td>13</td>
<td>15</td>
<td>4</td>
<td>26</td>
</tr>
</tbody>
</table>

sp_iqlmconfig procedure

Function

Controls license management configurations, displays and sets license type and status.

Syntax 1

```sql
sp_iqlmconfig 'edition', { 'SE' | 'SA' | 'EE' }
```

Table 7-41: Summary information for "edition" parameter

<table>
<thead>
<tr>
<th>Topic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>'EE' (Enterprise Edition)</td>
</tr>
<tr>
<td>Range of values</td>
<td>'SE' (Small Business)</td>
</tr>
<tr>
<td></td>
<td>'SA' (Single Application)</td>
</tr>
<tr>
<td></td>
<td>'EE' (Enterprise Edition)</td>
</tr>
<tr>
<td>Status</td>
<td>Static</td>
</tr>
</tbody>
</table>

Syntax 2

```sql
sp_iqlmconfig 'license type', { 'CP' | 'DT' | 'SF' | 'AC' | 'BC' | 'CH' | 'DH' | 'SH' | 'AH' | 'BH' }
```
Table 7-42: Summary information for "license type" parameter

<table>
<thead>
<tr>
<th>Topic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>'DT' (Development and Testing)</td>
</tr>
<tr>
<td>Range of values</td>
<td>'AC' (OEM CPU License)</td>
</tr>
<tr>
<td></td>
<td>'AH' (OEM CPU License Chip)</td>
</tr>
<tr>
<td></td>
<td>'BC' (OEM Standby License)</td>
</tr>
<tr>
<td></td>
<td>'BH' (OEM Standby License Chip)</td>
</tr>
<tr>
<td></td>
<td>'CP' (CPU License)</td>
</tr>
<tr>
<td></td>
<td>'CH' (CPU License Chip)</td>
</tr>
<tr>
<td></td>
<td>'DH' (Development and Testing License Chip)</td>
</tr>
<tr>
<td></td>
<td>'DT' (Development and Testing)</td>
</tr>
<tr>
<td></td>
<td>'EV' (Evaluation)</td>
</tr>
<tr>
<td></td>
<td>'SF' (Standby CPU License)</td>
</tr>
<tr>
<td></td>
<td>'SH' (Standby CPU License Chip)</td>
</tr>
<tr>
<td>Status</td>
<td>Static</td>
</tr>
</tbody>
</table>

Syntax 3
```
sp_iqlmconfig 'email severity', { 'ERROR' | 'WARNING' | 'INFORMATIONAL' | 'NONE' }
```

NONE designates that e-mail notification is disabled.

Syntax 4
```
sp_iqlmconfig 'smtp host', '<hostname>' |
```

<hostname> Specifies SMTP host used for e-mail notification.

Syntax 5
```
sp_iqlmconfig 'email sender', '<email address>' |
```

<email address> Specifies the sender’s e-mail address used for e-mail notification.

Syntax 6
```
sp_iqlmconfig 'email recipients', '<email recipients>' |
```

<email recipients> Specifies a comma-separated list of e-mail addresses to whom e-mail notifications will be sent.

Syntax 7
```
sp_iqlmconfig |
```

Permissions
DBA permissions necessary.

Usage
At startup, sp_iqlmconfig checks the edition type and license type of the license specified.

- If the specified license is not found, the server falls to grace mode.
- The specified license type becomes valid only when a non-null edition value is specified.
- If sp_iqlmconfig is called with no parameters (Syntax 3), it displays all the information above, as well as other information, such as:
System stored procedures

- Product Edition and License Type
- Which of the optional licenses are in use
- License count
- E-mail information
- General information about the license

sp_iqlocks procedure

Function
Shows information about locks in the database, for both the IQ store and the catalog store.

Syntax

```
sp_iqlocks ([connection], [owner:table_name] max_locks, [sort_order])
```

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage
Table 7-43 lists the optional sp_iqlocks parameters you can specify to restrict results.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>integer</td>
<td>Connection ID. With this option, the procedure returns information about locks for the specified connection only. Default is zero, which returns information about all connections.</td>
</tr>
<tr>
<td>owner:table_name</td>
<td>char (128)</td>
<td>Table name. With this option, the procedure returns information about locks for the specified table only. Default is NULL, which returns information about all tables in the database. If you do not specify owner, it is assumed that the caller of the procedure owns the table.</td>
</tr>
<tr>
<td>max_locks</td>
<td>integer</td>
<td>Maximum number of locks for which to return information. Default is 0, which returns all lock information.</td>
</tr>
<tr>
<td>sort_order</td>
<td>char(1)</td>
<td>Order in which to return information:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• C sorts by connection (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• T sorts by table_name</td>
</tr>
</tbody>
</table>
Description Displays information about current locks in the database. Depending on the options you specify, you can restrict results to show locks for a single connection, a single table, or a specified number of locks.

sp_iqlocks displays the following information, sorted as specified in the sort_order parameter:
### Table 7-44: sp_iqlocks columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conn_name</td>
<td>The name of the current connection.</td>
</tr>
<tr>
<td>conn_id</td>
<td>Connection ID that has the lock.</td>
</tr>
<tr>
<td>user_id</td>
<td>User associated with this connection ID.</td>
</tr>
<tr>
<td>table_type</td>
<td>The type of table. This type is either BASE for a table, GLBTMP for global temporary table, or MVIEW for a materialized view.</td>
</tr>
<tr>
<td>creator</td>
<td>The owner of the table.</td>
</tr>
<tr>
<td>table_name</td>
<td>Table on which the lock is held.</td>
</tr>
<tr>
<td>index_id</td>
<td>The index ID or NULL.</td>
</tr>
<tr>
<td>lock_class</td>
<td>String of characters indicating the type of lock:</td>
</tr>
<tr>
<td></td>
<td>S – share.</td>
</tr>
<tr>
<td></td>
<td>SW – share and write.</td>
</tr>
<tr>
<td></td>
<td>EW – exclusive and write.</td>
</tr>
<tr>
<td></td>
<td>E – exclusive.</td>
</tr>
<tr>
<td></td>
<td>P – phantom.</td>
</tr>
<tr>
<td></td>
<td>A – antiphantom.</td>
</tr>
<tr>
<td></td>
<td>W – write.</td>
</tr>
<tr>
<td></td>
<td>All locks listed have one of S, E, EW, or SW, and may also have P, A, or both. Phantom and antiphantom locks also have a qualifier of T or *:</td>
</tr>
<tr>
<td></td>
<td>T – the lock is with respect to a sequential scan.</td>
</tr>
<tr>
<td></td>
<td>* – the lock is with respect to all scans.</td>
</tr>
<tr>
<td></td>
<td>nnn – Index number; the lock is with respect to a particular index.</td>
</tr>
<tr>
<td>lock_type</td>
<td>Value identifying the lock (dependent on the lock class)</td>
</tr>
<tr>
<td>row_identifier</td>
<td>The identifier for the row or NULL.</td>
</tr>
</tbody>
</table>

If sp_iqlocks cannot find the connection ID or user name of the user who has a lock on a table, it displays a 0 (zero) for the connection ID and User unavailable for the user name.

**Note** Exclusive, phantom, or antiphantom locks can be placed on SQL Anywhere tables, but not on Sybase IQ tables. Unless you have explicitly taken out locks on a table in the catalog store, you never see these types of locks (or their qualifiers T, *, and nnn) in a Sybase IQ database. For information on how
locking works in SQL Anywhere tables, see *SQL Anywhere Server – SQL Usage*.

### Examples

The example shows the `sp_iqlocks` procedure call and its output in a Sybase IQ database. The procedure is called with all default options, so that the output shows all locks, sorted by connection.

```
call sp_iqlocks()
```

<table>
<thead>
<tr>
<th>conn_name</th>
<th>conn_id</th>
<th>user_id</th>
<th>table_type</th>
<th>creator</th>
<th>table_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>con1</td>
<td>70187172</td>
<td>'mary'</td>
<td>BASE</td>
<td>DBA</td>
<td>t1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index_id</th>
<th>lock_class</th>
<th>lock_duration</th>
<th>lock_type</th>
<th>row_identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIQ_IDX_T452_C19_FP</td>
<td>Table</td>
<td>Position</td>
<td>Table 1</td>
<td></td>
</tr>
</tbody>
</table>

### sp_iqmodifyadmin procedure

**Function**

Sets an option on a named login policy to a certain value. If no login policy is specified, the option is set on the root policy. In a multiplex, `sp_iqmodifyadmin` takes an optional parameter that is the multiplex server name.

**Syntax1**

```
call sp_iqmodifyadmin ('policy_option_name', 'value_in', ['login_policy_name'])
```

**Syntax2**

```
sp_iqmodifyadmin 'policy_option_name', 'value_in', 'login_policy_name'
```

**Syntax3**

```
sp_iqmodifyadmin policy_option_name, value_in, login_policy_name
```

**Syntax 4**

```
sp_iqmodifyadmin 'policy_option_name', 'value_in', 'login_policy_name', 'server_name'
```

**Usage**

- `policy_option_name` The login policy option to be changed.
- `value_in` New value for the login policy option.
- `login_policy_name` Name of the login policy whose login policy option needs to be changed.

**Permissions**

Requires DBA authority.

**See also**

“sp_iqpassword procedure” on page 441

ALTER LOGIN POLICY statement in Chapter 1, “SQL Statements,” in *Reference: Statements and Options.*
Examples

Example 1  Sets the login option locked to ON for the policy named lockeduser.

    call sp_iqmodifyadmin ('locked','on','lockeduser')

Example 2  Sets the login option locked to ON for the policy named lockeduser on the multiplex server named Writer1.

    call sp_iqmodifyadmin
       ('locked','on','lockeduser','Writer1')

sp_iqmodifylogin procedure

Function  Assigns a user to a login policy.
Syntax1  
    call sp_iqmodifylogin 'userid', ['login_policy_name']
Syntax2  
    sp_iqmodifylogin 'userid', ['login_policy_name']
Permissions  DBA authority required.
Usage  userid  Variable that holds the name of the account to modify.
        login_policy_name  (Optional) Specifies the name of the login policy to which the user will be assigned. If no login policy name is specified, the user is assigned to the root login policy.

Examples

Example 1  Assigns user joe to a login policy named expired_password:

    sp_iqmodifylogin 'joe', 'expired_password'

Example 2  Assigns user joe to the root login policy:

    call sp_iqmodifylogin ('joe')

See also  “sp_iqmodifyadmin procedure” on page 437

sp_iqobjectinfo procedure

Function  Returns partitions and dbspace assignments of database objects and sub-objects.
Syntax  
    sp_iqobjectinfo [ owner_name ] [, object_name ] [, object-type ]
Permissions  DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.
Usage

**owner_name**  Owner of the object. If specified, `sp_iqobjectinfo` displays output only for tables and join indexes with the specified owner. If not specified, `sp_iqobjectinfo` displays information on tables and join indexes for all users in the database.

**object_name**  Name of the table or join index. If not specified, `sp_iqobjectinfo` displays information on all tables and join indexes in the database.

**object-type**  Valid object types are `table` (the default) or `joinindex`.

If the object-type is a table, it must be enclosed in quotation marks.

All parameters are optional, and any parameter may be supplied independent of the value of another parameter.

Sybase recommends that you use input parameters with `sp_iqobjectinfo`; you can query the results of the `sp_iqobjectinfo` and it performs better if you use input parameters rather than using predicates in the `WHERE` clause of the query. For example, Query A is written as:

```sql
SELECT COUNT(*) FROM sp_iqobjectinfo()
WHERE owner = 'DBA'
AND object_name = 'tab_case510'
AND object_type = 'table'
AND sub_object_name is NULL
AND dbspace_name = 'iqmain7'
AND partition_name = 'P1'
```

Query B is Query A rewritten to use `sp_iqobjectinfo` input parameters:

```sql
SELECT COUNT(*) FROM sp_iqobjectinfo('DBA','tab_case510','table')
WHERE sub_object_name is NULL
AND dbspace_name = 'iqmain7'
AND PARTITION_NAME = 'P1'
```

Query B returns results faster than Query A. When the input parameters are passed to `sp_iqobjectinfo`, the procedure compares and joins fewer records in the system tables, thus doing less work compared to Query A. In Query B, the predicates are applied in the procedure itself and the procedure returns a smaller result set, so a smaller number of predicates is applied in the query.

The `sp_iqobjectinfo` stored procedure supports wildcard characters for interpreting `owner_name`, `object_name`, and `object_type`. It shows information for all dbspaces that match the given pattern in the same way the LIKE clause matches patterns inside queries.
System stored procedures

Description

Returns all the partitions and the dbspace assignments of a particular or all database objects (of type table and join index only) and its sub-objects. The sub-objects are columns, indexes, primary key, unique constraints, and foreign keys.

Table 7-45: sp_iqobjectinfo columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Name of the owner of the object.</td>
</tr>
<tr>
<td>object_name</td>
<td>Name of the object (of type table and join index only) located on the dbspace.</td>
</tr>
<tr>
<td>sub_object_name</td>
<td>Name of the object located on the dbspace.</td>
</tr>
<tr>
<td>object_type</td>
<td>Type of the object (column, index, primary key, unique constraint, foreign key, partition, join index or table).</td>
</tr>
<tr>
<td>object_id</td>
<td>Global object id of the object.</td>
</tr>
<tr>
<td>id</td>
<td>Table id or join-index id of the object.</td>
</tr>
<tr>
<td>dbspace_name</td>
<td>Name of the dbspace on which the object resides. The string “[multiple]” is displayed for a special meta row for partitioned objects. The [multiple] row indicates that multiple rows follow in the output to describe the table or column.</td>
</tr>
<tr>
<td>partition_name</td>
<td>Name of the partition for the given object.</td>
</tr>
</tbody>
</table>

Examples

Displays information about partitions and dbspace assignments of a specific database object and sub-objects owned by a specific user:

```sql
sp_iqobjectinfo GROUP0,Departments
```

<table>
<thead>
<tr>
<th>owner</th>
<th>object_name</th>
<th>sub_object_name</th>
<th>object_type</th>
<th>object_id</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP0</td>
<td>Departments</td>
<td>(NULL)</td>
<td>table</td>
<td>3632</td>
<td>738</td>
</tr>
<tr>
<td>GROUP0</td>
<td>Departments</td>
<td>DepartmentID</td>
<td>column</td>
<td>3633</td>
<td>738</td>
</tr>
<tr>
<td>GROUP0</td>
<td>Departments</td>
<td>DepartmentName</td>
<td>column</td>
<td>3634</td>
<td>738</td>
</tr>
<tr>
<td>GROUP0</td>
<td>Departments</td>
<td>DepartmentHeadID</td>
<td>column</td>
<td>3635</td>
<td>738</td>
</tr>
<tr>
<td>GROUP0</td>
<td>Departments</td>
<td>DepartmentsKey</td>
<td>primary key</td>
<td>83</td>
<td>738</td>
</tr>
<tr>
<td>GROUP0</td>
<td>Departments</td>
<td>PK_DepartmentHeadID_EmployeeID</td>
<td>foreign key</td>
<td>92</td>
<td>738</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dbspace_name</th>
<th>partition_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>iq_main</td>
<td>(NULL)</td>
</tr>
<tr>
<td>iq_main</td>
<td>(NULL)</td>
</tr>
<tr>
<td>iq_main</td>
<td>(NULL)</td>
</tr>
<tr>
<td>iq_main</td>
<td>(NULL)</td>
</tr>
<tr>
<td>iq_main</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

Displays information about partitions and dbspace assignments of a specific database object and sub-objects owned by a specific user for object-type table:

```sql
sp_iqobjectinfo DBA,sale,'table'
```
### sp_iqpassword procedure

**Function**
Changes a user’s password. The preferred way to create a user is by using the CREATE USER statement. See “CREATE USER statement,” in Chapter 1, “SQL Statements,” in Reference: Statements and Options.

**Syntax1**
```sql
call sp_iqpassword ('caller_password', 'new_password' [, 'user_name'])
```

**Syntax2**
```sql
sp_iqpassword 'caller_password', 'new_password' [, 'user_name']
```

**Permissions**
None to set your own password; DBA authority required to set other users’ passwords.

**Usage**
- **caller_password** Your password. When you are changing your own password, this is your old password. When the DBA is changing another user’s password, caller_password is the DBA’s password.
- **new_password** New password for the user, or for loginnname.
- **user_name** Login name of the user whose password is being changed by the DBA. Do not specify user_name when changing your own password.

**Description**
Any user can change his or her own password using sp_iqpassword. The DBA can change the password of any existing user.

**Examples**

**Example 1** Changes the password of the logged in user from irk103 to exP984:
```sql
sp_iqpassword 'irk103', 'exP984'
```

**Example 2** Changes the password of user joe from eprr45 to pdi032 only if the logged in user has DBA privileges or the user is joe himself:
```sql
call sp_iqpassword ('eprr45', 'pdi032', 'joe')
```
System stored procedures

**sp_iqpkeys procedure**

**Function**
Displays information about primary keys and primary key constraints by table, column, table owner, or for all Sybase IQ tables in the database.

**Syntax**
```
sp_iqpkeys {{ [table-name] }, [column-name], [table-owner] }
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**
- **table-name** The name of a base or global temporary table. If specified, the procedure returns information about primary keys defined on the specified table only.
- **column-name** The name of a column. If specified, the procedure returns information about primary keys on the specified column only.
- **table-owner** The owner of a table or table. If specified, the procedure returns information about primary keys on tables owned by the specified owner only.

One or more of the parameters can be specified. If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. If none of the parameters are specified, a description of all primary keys on all tables in the database is displayed. If any of the specified parameters is invalid, no rows are displayed in the output.

**Table 7-46: sp_iqpkeys usage examples**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_iqpkeys sales</td>
<td>Displays information about primary keys defined on table sales</td>
</tr>
<tr>
<td>sp_iqpkeys sales, NULL, DBA</td>
<td>Displays information about primary keys defined on table sales owned by DBA</td>
</tr>
<tr>
<td>sp_iqpkeys sales, store_id, DBA</td>
<td>Displays information about primary key defined on column store_id of table sales owned by DBA</td>
</tr>
<tr>
<td>sp_iqpkeys NULL, NULL, DBA</td>
<td>Displays information about primary keys defined on all tables owned by DBA</td>
</tr>
</tbody>
</table>

**Description**
The sp_iqpkeys stored procedure displays the following information about primary keys on base and global temporary tables in a database:
**Table 7-47: sp_iqpkeys columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table</td>
</tr>
<tr>
<td>table_owner</td>
<td>The owner of the table</td>
</tr>
<tr>
<td>column_name</td>
<td>The name of the column(s) on which the primary key is defined</td>
</tr>
<tr>
<td>column_id</td>
<td>The column ID</td>
</tr>
<tr>
<td>constraint_name</td>
<td>The name of the primary key constraint</td>
</tr>
<tr>
<td>constraint_id</td>
<td>The primary key constraint ID</td>
</tr>
</tbody>
</table>

**Note** The sp_iqpkeys stored procedure exists only in databases created with Sybase IQ version 12.6 or later.

**Examples**

Display the primary keys defined on columns of table `sales1`:

```
sp_iqpkeys sales1
```

```
table_name table_owner column_name column_id constraint_name constraint_id
sales1 DBA store_id 1 MA114 114
```

Display the primary keys defined on columns of table `sales2`:

```
sp_iqpkeys sales2
```

```
table_name table_owner column_name column_id constraint_name constraint_id
sales2 DBA store_id,1,2 MA115 115
```

Display the primary keys defined on the column `store_id` of table `sales2`:

```
sp_iqpkeys sales2, store_id
```

```
table_name table_owner column_name column_id constraint_name constraint_id
sales2 DBA store_id 1 MA115 115
```

**See also**

“sp_iqindex and sp_iqindex_alt procedures” on page 415

“sp_iqcolumn procedure” on page 370

**sp_iqprocedure procedure**

**Function**

Displays information about system and user-defined procedures.

**Syntax**

```
sp_iqprocedure [ proc-name ], [ proc-owner ], [ proc-type ]
```
Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

proc-name  The name of the procedure.
proc-owner  The owner of the procedure.
proc-type   The type of procedure. Allowed values are:

- SYSTEM: displays information about system procedures (procedures owned by user SYS or dbo) only
- ALL: displays information about user and system procedures
- Any other value: displays information about user procedures

The sp_iqprocedure procedure can be invoked without any parameters. If no parameters are specified, only information about user-defined procedures (procedures not owned by dbo or SYS) is displayed by default.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, sp_iqprocedure NULL, NULL, SYSTEM and sp_iqprocedure NULL, user1.
The `sp_iqprocedure` stored procedure displays information about procedures in a database. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if `proc-name` is specified, only information about the specified procedure is displayed. If `proc-owner` is specified, `sp_iqprocedure` returns only information about procedures owned by the specified owner. If no parameters are specified, `sp_iqprocedure` displays information about all the user-defined procedures in the database.

The `sp_iqprocedure` procedure returns information in the following columns:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sp_iqprocedure</code></td>
<td>Displays information about all procedures in the database not owned by <code>dbo</code> or <code>SYS</code></td>
</tr>
<tr>
<td><code>sp_iqprocedure sp_test</code></td>
<td>Displays information about the procedure <code>sp_test</code></td>
</tr>
<tr>
<td><code>sp_iqprocedure non_existing_proc</code></td>
<td>No rows returned, as the procedure <code>non_existing_proc</code> does not exist</td>
</tr>
<tr>
<td><code>sp_iqprocedure NULL, DBA</code></td>
<td>Displays information about all procedures owned by <code>DBA</code></td>
</tr>
<tr>
<td><code>sp_iqprocedure sp_test, DBA</code></td>
<td>Displays information about the procedure <code>sp_test</code> owned by <code>DBA</code></td>
</tr>
<tr>
<td><code>sp_iqprocedure sp_iqtable</code></td>
<td>The procedure <code>sp_iqtable</code> is not a system procedure. If there is no user-defined procedure also named <code>sp_iqtable</code>, no rows are returned. (By default only user-defined procedures are returned.)</td>
</tr>
<tr>
<td><code>sp_iqprocedure sp_iqtable, dbo</code></td>
<td>No rows returned, as the procedure <code>sp_iqtable</code> is not a user procedure (by default only user procedures returned)</td>
</tr>
<tr>
<td><code>sp_iqprocedure NULL, SYSTEM</code></td>
<td>Displays information about all system procedures (owned by <code>dbo</code> or <code>SYS</code>)</td>
</tr>
<tr>
<td><code>sp_iqprocedure sp_iqtable, NULL, 'SYSTEM'</code></td>
<td>Displays information about the system procedure <code>sp_iqtable</code></td>
</tr>
<tr>
<td><code>sp_iqprocedure sp_iqtable, dbo, ALL</code></td>
<td>Displays information about the system procedure <code>sp_iqtable</code> owned by <code>dbo</code></td>
</tr>
</tbody>
</table>

**Description**

The `sp_iqprocedure` stored procedure displays information about procedures in a database. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if `proc-name` is specified, only information about the specified procedure is displayed. If `proc-owner` is specified, `sp_iqprocedure` returns only information about procedures owned by the specified owner. If no parameters are specified, `sp_iqprocedure` displays information about all the user-defined procedures in the database.

The `sp_iqprocedure` procedure returns information in the following columns:
System stored procedures

Table 7-49: sp_iqprocedure columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc_name</td>
<td>The name of the procedure</td>
</tr>
<tr>
<td>proc_owner</td>
<td>The owner of the procedure</td>
</tr>
<tr>
<td>proc_defn</td>
<td>The command used to create the procedure. For hidden procedures, the keyword ‘HIDDEN’ is displayed.</td>
</tr>
<tr>
<td>replicate</td>
<td>Displays Y if the procedure is a primary data source in a Replication Server installation; N if not.</td>
</tr>
<tr>
<td>srvid</td>
<td>Indicates the remote server, if the procedure is on a remote database server</td>
</tr>
<tr>
<td>remarks</td>
<td>A comment string</td>
</tr>
</tbody>
</table>

Examples

Displays information about the user-defined procedure sp_test:

```sql
sp_iqprocedure sp_test
```

```
<table>
<thead>
<tr>
<th>proc_name</th>
<th>proc_owner</th>
<th>proc_defn</th>
<th>replicate</th>
<th>srvid</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_test</td>
<td>DBA</td>
<td>create procedure N (NULL) (NULL)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBA.sp_test(in n1 integer) begin message 'sp_test' end</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Displays information about all procedures owned by user DBA:

```sql
sp_iqprocedure NULL, DBA
```

```
<table>
<thead>
<tr>
<th>proc_name</th>
<th>proc_owner</th>
<th>proc_defn</th>
<th>replicate</th>
<th>srvid</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_test</td>
<td>DBA</td>
<td>create procedure N (NULL) (NULL)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBA.sp_test(in n1 integer) begin message 'sp_test' end</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp_dept</td>
<td>DBA</td>
<td>create procedure N (NULL) (NULL)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBA.sp_dept() begin end</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

See also

“CREATE USER statement,” in Chapter 1, “SQL Statements,” in Reference: Statements and Options

sp_iqprocparm procedure

Function
Displays information about stored procedure parameters, including result set variables and SQLSTATE/SQLCODE error values.

Syntax

```
sp_iqprocparm [ proc-name ], [ proc-owner ], [ proc-type ]
```
Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Usage

proc-name  The name of the procedure.

proc-owner  The owner of the procedure.

proc-type  The type of procedure. Allowed values are:

- SYSTEM: displays information about system procedures (procedures owned by user SYS or dbo) only
- ALL: displays information about user and system procedures
- Any other value: displays information about user procedures

You can invoke sp_iqprocparm without parameters. If you do not specify any parameters, input/output and result parameters of user-defined procedures (procedures not owned by dbo or SYS) appear.

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, sp_iqprocparm NULL, NULL, SYSTEM and sp_iqprocparm NULL, user1.
Table 7-50: sp_iqprocparm usage examples

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_iqprocparm</td>
<td>Displays parameters for all procedures in the database not owned by dbo or SYS</td>
</tr>
<tr>
<td>sp_iqprocparm sp_test</td>
<td>Displays information about the procedure sp_test</td>
</tr>
<tr>
<td>non_existing_proc</td>
<td>No rows returned, as the procedure non_existing_proc does not exist</td>
</tr>
<tr>
<td>sp_iqprocparm NULL, DBA</td>
<td>Displays parameters for all procedures owned by DBA</td>
</tr>
<tr>
<td>sp_iqprocparm sp_test, DBA</td>
<td>Displays parameters for the procedure sp_test owned by DBA</td>
</tr>
<tr>
<td>sp_iqprocparm sp_iqtable</td>
<td>sp_iqtable is a system procedure. If there is no user-defined procedure also named sp_iqtable, no rows are returned. (By default, only user-defined procedures are returned.)</td>
</tr>
<tr>
<td>sp_iqprocparm sp_iqtable, dbo</td>
<td>No rows returned, as the procedure sp_iqtable is not a user procedure. (By default, only user procedures are returned.)</td>
</tr>
<tr>
<td>sp_iqprocparm NULL, NULL, SYSTEM</td>
<td>Displays parameters for all system procedures (owned by dbo or SYS)</td>
</tr>
<tr>
<td>sp_iqprocparm sp_iqtable, NULL, SYSTEM</td>
<td>Displays parameters of the system procedure sp_iqtable</td>
</tr>
<tr>
<td>sp_iqprocparm sp_iqtable, dbo, ALL</td>
<td>Displays parameters of the system procedure sp_iqtable owned by dbo</td>
</tr>
</tbody>
</table>

Description

The sp_iqprocparm stored procedure displays information about stored procedure parameters, including result set variables and SQLSTATE/SQLCODE error values. If you specify one or more parameters, the result is filtered by the specified parameters. For example, if proc-name is specified, only information about parameters to the specified procedure displays. If proc-owner is specified, sp_iqprocparm only returns information about parameters to procedures owned by the specified owner. If no parameters are specified, sp_iqprocparm displays information about parameters to all the user-defined procedures in the database.

The sp_iqprocparm procedure returns information in the following columns:
### Table 7-51: sp_iqprocparm columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc_name</td>
<td>The name of the procedure</td>
</tr>
<tr>
<td>proc_owner</td>
<td>The owner of the procedure</td>
</tr>
<tr>
<td>parm_name</td>
<td>The name of the parameter</td>
</tr>
</tbody>
</table>
| parm_type   | The type of parameter is one of the following values:  
|             |   • normal parameter (variable)  
|             |   • result variable: used with procedures that return result sets  
|             |   • SQLSTATE error value  
|             |   • SQLCODE error value          |
| parm_mode   | The mode of the parameter: whether a parameter supplies a value to the procedure, returns a value, does both, or does neither. Parameter mode is one of the following:  
|             |   • in: parameter supplies a value to the procedure  
|             |   • out: parameter returns a value  
|             |   • inout: parameter supplies as well as returns a value  
|             |   • NULL: parameter neither supplies nor returns a value |
| domain_name | The name of the data type of the parameter as listed in the SYSDOMAIN system table |
| width       | The length of string parameters, the precision of numeric parameters, and the number of bytes of storage for all other data types |
| scale       | The number of digits after the decimal point for numeric data type parameters and zero for all other data types |
| default     | The default value of the parameter, held as a string |

**Examples**

Display information about the parameters of the user-defined procedure `sp_test`:

```sql
sp_iqprocparm sp_test
proc_name proc_owner parm_name parm_type parm_mode domain_name width scale default
sp_test DBA ID normal in integer 4 0 (NULL)
```

Display information about the parameters of the system procedure `sp_iqshowcompression`:

```sql
sp_iqprocparm sp_iqshowcompression, dbo, system
proc_name proc_owner parm_name parm_type parm_mode domain_name width scale default
```
System stored procedures

sp_iqshowcompression dbo @owner_name normal in
char 128 0 (NULL)
sp_iqshowcompression dbo @table_name normal in
char 128 0 (NULL)
sp_iqshowcompression dbo @column_name normal in
char 128 0 (NULL)
sp_iqshowcompression dbo Column result out
char 128 0 (NULL)
sp_iqshowcompression dbo Compression result out
char 3 0 (NULL)

See also

CREATE PROCEDURE statement in Chapter 1, “SQL Statements,” in Reference: Statements and Options

sp_iqrebuildindex procedure

Function
Rebuilds one or more indexes on a table with the original IQ UNIQUE value specified in the CREATE TABLE statement, or a new IQ UNIQUE value to change storage required and/or query performance. To rebuild an index other than the default index, specify the index name.

Syntax
sp_iqrebuildindex (table_name, index_clause)

Permissions
You must have INSERT permission on a table to rebuild an index on that table.

Usage

table_name Partial or fully qualified table name on which the index rebuild process takes place. If the user both owns the table and executes the procedure, a partially qualified name may be used; otherwise, the table name must be fully qualified.

index_clause One or more of the following strings, separated by spaces:
column column_name [count]
[dbspace dbspace_name]
index index_name
[dbspace dbspace_name]

Each column_name or index_name must refer to a column or index on the specified table. If you specify a column_name or index_name multiple times, the procedure returns an error and no index is rebuilt.
The *count* is a nonnegative number that represents the IQ UNIQUE value. In a
CREATE TABLE statement, IQ UNIQUE (count) approximates how many
distinct values can be in a given column. The number of distinct values affects
query speed and storage requirements. For details, see “Optimizing storage
and query performance,” in Chapter 5, “Working with Database Objects” in
the System Administration Guide: Volume 1.

You must specify the keywords column and index. The keyword dbspace is
optional. These keywords are not case-sensitive.

Sybase IQ rebuilds the column or index in the same dbspace where the original
resided unless you specify dbspace db-space-name.

See also

“sp_iqindexfragmentation procedure” on page 419, and “sp_iqrowdensity
procedure” on page 456.

Chapter 6, “Using Sybase IQ Indexes” in System Administration Guide:
Volume 1.

“FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in

Description

If you specify a column name, the procedure rebuilds the default index for that
column, and no index name is needed. Specifying the name of the default index
assigned by Sybase IQ in addition to the column name in this situation returns
an error. If you omit *count* after the *column_name*, value 0 (zero) is used as the
default.

If the default index is a one-byte index, sp_iqrebuildindex always rebuilds it as
a one-byte index no matter what IQ UNIQUE value the user specified.

For one-byte default indexes, if the specified value in *column_name (count)* is
0 or greater than 256, the column’s cardinality value is used to update the
approx_unique_count column in SYS.SYSIQCOLUMN.

If the column has the data type VARCHAR or VARBINARY greater than 255
bytes, sp_iqrebuildindex will not rebuild a default index.

sp_iqrebuildindex rebuilds a WD index on a column of data type LONG
VARCHAR (CLOB).

If the default index is a two-byte index, and the specified count is 0 or greater
than 65536, the column’s cardinality value determines whether to rebuild the
default into a one-byte or two-byte index, and that value is used to update the
approx_unique_count column in SYS.SYSIQCOLUMN.

If you specify a nonzero IQ UNIQUE value, the default index is rebuilt as a one-
byte, two-byte, or flat default index, with exceptions described above.
If you specify an IQ UNIQUE value of zero or no IQ UNIQUE value, the MINIMIZE_STORAGE option controls how the index is rebuilt:

- If MINIMIZE_STORAGE option is set ON, the index is rebuilt as a one-byte default index first, and converted to two-byte or flat if necessary.
- If MINIMIZE_STORAGE is set OFF, the index is rebuilt using the default for the data type. For more information, see “Sybase IQ index types” Chapter 7, “System Procedures” in the System Administration Guide: Volume 1.

Examples

Rebuilds the default index on column Surname:

```
sp_iqrebuildindex 'empl1', 'column dept_id'
```

or:

```
call sp_iqrebuildindex ('empl1', 'column dept_id')
```

Creates a flat default index on column cl:

```CREATE TABLE mytable (cl int IQ UNIQUE 1000000000)```

Converts the default one-byte index to a two-byte index:

```
sp_iqrebuildindex 'mytable', 'column cl 1024'
```

or:

```
call sp_iqrebuildindex ('mytable', 'column cl 1024')
```

See also

“sp_iqindexfragmentation procedure” on page 419, and “sp_iqrowdensity procedure” on page 456.

“FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in Reference: Statements and Options.


**sp_iqrename procedure**

**Function**

Renames user-created tables, columns, indexes, constraints (unique, primary key, foreign key, and check), stored procedures, and functions.

**Syntax**

```
sp_iqrename object-name, new-name [ , object-type ]
```

**Permissions**

Must be the owner of the table or have DBA authority or alter permission on the object. Requires exclusive access to the object.

**Usage**

```
object-name  The original name of the user-created object.
```
Optionally, owner-name can be specified as part of object-name as owner-name.object-name, where owner-name is the name of the owner of the object being renamed. If owner-name is not specified, the user calling sp_iqrename is assumed to be the owner of the object. Note that the object is successfully renamed only if the user calling sp_iqrename has the required permissions to rename the object.

If the object to be renamed is a column, index, or constraint, you must specify the name of the table with which the object is associated. For a column, index, or constraint, object-name can be of the form table-name.object-name or owner-name.table-name.object-name.

new-name The new name of the object. The name must conform to the rules for identifiers and must be unique for the type of object being renamed.

object-type An optional parameter that specifies the type of the user-created object being renamed, that is, the type of the object object-name. The object-type parameter can be specified in either upper or lowercase.

<table>
<thead>
<tr>
<th>object-type parameter</th>
<th>Specifies</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>The object being renamed is a column</td>
</tr>
<tr>
<td>index</td>
<td>The object being renamed is an index</td>
</tr>
<tr>
<td>constraint</td>
<td>The object being renamed is a unique, primary key, check, or referential (foreign key) constraint</td>
</tr>
<tr>
<td>procedure</td>
<td>The object being renamed is a function</td>
</tr>
<tr>
<td>object-type not specified</td>
<td>The object being renamed is a table</td>
</tr>
</tbody>
</table>

Warning! You must change appropriately the definition of any dependent object (procedures, functions, and views) on an object being renamed by sp_iqrename. The sp_iqrename procedure does not automatically update the definitions of dependent objects. You must change these definitions manually.

The sp_iqrename stored procedure renames user-created tables, columns, indexes, constraints (unique, primary key, foreign key, and check), and functions.

If you attempt to rename an object with a name that is not unique for that type of object, sp_iqrename returns the message “Item already exists.”

sp_iqrename does not support renaming a view, a procedure, an event or a data type. The message “Feature not supported.” is returned by sp_iqrename, if you specify event or datatype as the object-type parameter.
System stored procedures

Examples

Renames the table titles owned by user shweta to books:

```
sp_iqrename shweta.titles, books
```

Renames the column id of the table books to isbn:

```
sp_iqrename shweta.books.id, isbn, column
```

Renames the index idindex on the table books to isbnindex:

```
sp_iqrename books.idindex, isbnindex, index
```

Renames the primary key constraint prim_id on the table books to prim_isbn:

```
sp_iqrename books.prim_id, prim_isbn, constraint
```

See also

ALTER TABLE statement RENAME clause and ALTER INDEX statement RENAME clause in Reference: Statements and Options

sp_iq_reset_identity procedure

Function

Sets the seed of the Identity/Autoincrement column associated with the specified table to the specified value.

Syntax

```
sp_iq_reset_identity (table_name, table_owner, value)
```

Usage

Syntax You must specify table_name, table_owner, and value.

Permissions

None required.

Description

The Identity/Autoincrement column stores a number that is automatically generated. The values generated are unique identifiers for incoming data. The values are sequential, are generated automatically, and are never reused, even when rows are deleted from the table. The seed value specified replaces the default seed value and persists across database shutdowns and failures.

Example

The following example creates an Identity column with a starting seed of 50:

```
CREATE TABLE mytable(c1 INT identity)
call sp_iq_reset_identity('mytable', 'dba', 50)
```

See also

“sp_iqcolumn procedure” on page 370

sp_iqrestoreaction procedure

Function

Shows what restore actions are needed to bring database to a consistent state with a given past date.

Syntax

```
sp_iqrestoreaction [ timestamp ]
```

454 Sybase IQ
Parameters

**timestamp**  Specifies the past date target.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

sp_iqrestoreaction returns an error if the database cannot be brought to a consistent state for the timestamp. Otherwise, suggests restore actions that will return the database to a consistent state.

The common point to which the database can be restored coincides with the last backup time that backed up read-write files just before the specified timestamp. The backup may be all-inclusive or read-write files only.

Output may not be in exact ascending order based on backup time. If a backup archive consists of multiple read-only dbfiles, it may contain multiple rows (with the same backup time and backup id).

If you back up a read-only dbfile or dbspace multiple times, the restore uses the last backup. The corresponding backup time could be after the specified timestamp, as long as the dbspace/dbfile alter ID matches the dbspace/dbfile alter ID recorded in the last read-write backup that is restored.

sp_iqrestoreaction returns the following:

**Table 7-53: sp_iqrestoreaction columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence_number</td>
<td>Orders the steps to be taken</td>
</tr>
<tr>
<td>backup_id</td>
<td>Identifier for the backup transaction</td>
</tr>
<tr>
<td>backup_archive_list</td>
<td>List of archive files in the backup</td>
</tr>
<tr>
<td>backup_time</td>
<td>Time of the backup taken</td>
</tr>
<tr>
<td>virtual_type</td>
<td>Type of virtual backup: “Non-virtual,” “Decoupled,” or “Encapsulated”</td>
</tr>
<tr>
<td>restore_dbspace</td>
<td>Can be empty. Indicates that all dbspaces are to be restored from the backup archive</td>
</tr>
<tr>
<td>restore_dbfile</td>
<td>Could be empty. Indicates that all dbfiles in the given dbspace are to be restored from the backup archive</td>
</tr>
<tr>
<td>backup_comment</td>
<td>User comment</td>
</tr>
</tbody>
</table>

**Example**

Sample output of sp_iqrestoreaction:

<table>
<thead>
<tr>
<th>sequence_number</th>
<th>backup_id</th>
<th>backup_archive_list</th>
<th>backup_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1192</td>
<td>c:\temp\b1</td>
<td>2008-09-23 14:47:40.0</td>
</tr>
<tr>
<td>2</td>
<td>1201</td>
<td>c:\temp\b2.inc</td>
<td>2008-09-23 14:48:05.0l</td>
</tr>
<tr>
<td>3</td>
<td>1208</td>
<td>c:\temp\b3.inc</td>
<td>2008-09-23 14:48:13.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual_type</th>
<th>restore_dbspace</th>
<th>restore_dbfile</th>
<th>backup_comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonvirtual</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**sp_iqrowdensity procedure**

**Function**
Reports information about the internal row fragmentation for a table at the FP index level.

**Syntax**
```
    dbo.sp_iqrowdensity ('target')
    target:(table table-name | (column column-name (...))
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**
- `table-name`  Target table `table-name` reports on all columns in the named table.
- `column-name` Target column `column-name` reports on the named column in the target table. You may specify multiple target columns, but must repeat the keyword each time.

You must specify the keywords `table` and `column`. These keywords are not case-sensitive.

**Description**
sp_iqrowdensity measures row fragmentation at the default index level. Density is the ratio of the minimum number of pages required by an index for existing table rows to the number of pages actually used by the index. This procedure returns density as a number such that $0 < \text{density} \leq 1$. For example, if an index that requires 8 pages minimum storage occupies 10 pages, its density is .8.

The density reported does not indicate the number of disk pages that may be reclaimed by re-creating or reorganizing the default index.

This procedure displays information about the row density of a column, but does not recommend further action. You must determine whether or not to re-create, reorganize, or rebuild an index.

**Example**
Reports the row density on column `ID` in table `SalesOrders`:
```
    sp_iqrowdensity('column grupo.SalesOrders.ID')
```

<table>
<thead>
<tr>
<th>Tablename</th>
<th>ColumnName</th>
<th>IndexType</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>'GROUPO.SalesOrders'</td>
<td>'ID'</td>
<td>'Flat style FP'</td>
<td>'1.0'</td>
</tr>
</tbody>
</table>

**See also**
“FP_LOOKUP_SIZE option” and “MINIMIZE_STORAGE option” in Chapter 2, “Database Options” in Reference: Statements and Options.

**sp_iqshowpsexe procedure**

**Function**
Displays information about the settings of database options that control the priority of tasks and resource usage for connections.

**Syntax**
```
sp_iqshowpsexe [ connection-id ]
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**
- **connection-id** An integer representing the connection ID.

If `connection-id` is specified, `sp_iqshowpsexe` returns information only about the specified connection. If `connection-id` is not specified, `sp_iqshowpsexe` returns information about all connections.

If the specified `connection-id` does not exist, `sp_iqshowpsexe` returns no rows.

**Description**
The `sp_iqshowpsexe` stored procedure displays information about the settings of database options that control the priority of tasks and resource usage for connections, which is useful to database administrators for performance tuning.
System stored procedures

### Table 7-54: `sp_iqshowpsexe` columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectionid</td>
<td>The connection ID</td>
</tr>
</tbody>
</table>
| application      | Information about the client application that opened the connection. Includes the AppInfo connection property information:  
                  | HOST: the host name of the client machine                                                      |
|                  | EXE: the name of the client executable (Windows only)                                          |
|                  | APPINFO: the APPINFO in the client connection string, if specified                             |
| userid           | Login name of the user that opened the connection                                               |
| iqgovern_priority| Value of the database option `IQGOVERN_PRIORITY` that assigns a priority to each query waiting in the -iqgovern queue. By default, this option has a value of 2 (MEDIUM). The values 1, 2, and 3 are shown as HIGH, MEDIUM, and LOW, respectively. |
| max_query_time   | Value of the database option `MAX_QUERY_TIME` that sets a limit, so that the optimizer can disallow very long queries. By default, this option is disabled and has a value of 0. |
| query_row_limit  | Value of the database option `QUERY_ROWS_RETURNED_LIMIT` that sets the row threshold for rejecting queries based on the estimated size of the result set. The default is 0, which means there is no limit. |
| query_temp_space_limit | Value of the database option `QUERY_TEMP_SPACE_LIMIT` (in MB) that constrains the use of temporary IQ dbspace by user queries. The default value is 2000MB. |
| max_cursors      | Value of the database option `MAX_CURSOR_COUNT` that specifies a resource governor to limit the maximum number of cursors a connection can use at once. The default value is 50. A value of 0 implies no limit. |
| max_statements   | Value of the database option `MAX_STATEMENT_COUNT` that specifies a resource governor to limit the maximum number of prepared statements that a connection can use at once. The default value is 100. A value of 0 implies no limit. |

**Note** The AppInfo property may not be available from Open Client or jConnect applications such as the Java version of Interactive SQL (dbisql) or Sybase Central. If the AppInfo property is not available, the application column is blank.
Example

Display information about the settings of database options that control the priority of tasks and resource usage for connection ID 2:

```
sp_iqshowpsexe 2
```

```
connectionid 2
   application
   HOST=GOODGUY-XP;EXE=C:\Program Files\Sybase\IQ-15_1\bin32\dbisqlg.exe;

userid  iqgovern_priority max_query_time query_row_limit
   DBA MEDIUM 0 0
query_temp_space_limit max_statements max_cursors
   2000 50 100
```

See also

In Chapter 7, “System Procedures”: “sp_iqconnection procedure” on page 373, “sp_iqcontext procedure” on page 378, and “sa_conn_info system procedure” on page 494

“CONNECTION_PROPERTY function [System]” on page 139

In Chapter 2, “Database Options” in Reference: Statements and Options: IQGOVERN_MAX_PRIORITY option, IQGOVERN_PRIORITY option, IQGOVERN_PRIORITY_TIME option, MAX_QUERY_TIME option, QUERY_ROWS_RETURNED_LIMIT option, QUERY_TEMP_SPACE_LIMIT option, MAX_CURSOR_COUNT option, and MAX_STATEMENT_COUNT option

“AppInfo connection parameter [App]” in Chapter 4, “Connection and Communication Parameters” in the System Administration Guide: Volume 1

**sp_iqspaceinfo procedure**

Function

Displays the number of blocks used by each object in the current database and the name of the dbspace in which the object is located.

Syntax

```
sp_iqspaceinfo ['main']
 | [(table table-name | index index-name) [...]]
```

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

For the current database, displays the object name, number of blocks used by each object, and the name of the dbspace. sp_iqspaceinfo requires no parameters.

The information returned by sp_iqspaceinfo is helpful in managing dbspaces.
The following output is from the `sp_iqspaceinfo` stored procedure run on the `iqdemo` database. Output for some tables and indexes have been removed in this example.

<table>
<thead>
<tr>
<th>Name</th>
<th>NBlocks</th>
<th>dbspace_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contacts</td>
<td>19</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
<tr>
<td>SalesOrderItems.DBA.ASIQ_IDX_T205_C5_FP</td>
<td>56</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
<tr>
<td>Contacts.DBA.ASIQ_IDX_T206_C10_FP</td>
<td>55</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
<tr>
<td>Contacts.DBA.ASIQ_IDX_T206_C1_FP</td>
<td>61</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
<tr>
<td>Contacts.DBA.ASIQ_IDX_T206_C9_FP</td>
<td>55</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
<tr>
<td>Contacts.DBA.ASIQ_IDX_T206_I11_HG</td>
<td>19</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
<tr>
<td>Customers</td>
<td>20</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
<tr>
<td>Customers.DBA.ASIQ_IDX_T207_C1_FP</td>
<td>61</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
<tr>
<td>Customers.DBA.ASIQ_IDX_T207_C2_FP</td>
<td>55</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
<tr>
<td>Customers.DBA.ASIQ_IDX_T207_I10_HG</td>
<td>19</td>
<td>IQ_SYSTEM_MAIN</td>
</tr>
</tbody>
</table>

See also

“sp_iqindexinfo procedure” on page 421, “sp_iqdbspace procedure” on page 388, and “sp_iqdbspaceinfo procedure” on page 391

Chapter 5, “Working with Database Objects” in the System Administration Guide: Volume 1

### sp_iqspaceused procedure

**Function**

Shows information about space available and space used in the IQ store and IQ temporary store.

**Syntax**

```
sp_iqspaceused(out mainKB unsigned bigint,
               out mainKBUsed unsigned bigint,
               out tempKB unsigned bigint,
               out tempKBUsed unsigned bigint)
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**

`sp_iqspaceused` returns four values as unsigned bigint out parameters. This system stored procedure can be called by user-defined stored procedures to determine the amount of main and temporary IQ store space in use.

**Description**

`sp_iqspaceused` returns a subset of the information provided by `sp_iqstatus`, but allows the user to return the information in SQL variables to be used in calculations.
Example

sp_iqspaceused requires four output parameters. The following example shows the creation of a user-defined stored procedure `myspace` that declares the four output parameters and then calls `sp_iqspaceused`:

```sql
create procedure dbo.myspace()
begin
    declare mt unsigned bigint;
    declare mu unsigned bigint;
    declare tt unsigned bigint;
    declare tu unsigned bigint;
    call sp_iqspaceused(mt,mu,tt,tu);
    select cast(mt/1024 as unsigned bigint) as mainMB,
        cast(mu/1024 as unsigned bigint) as mainusedMB,
        mu*100/mt as mainPerCent,
        cast(tt/1024 as unsigned bigint) as tempMB,
        cast(tu/1024 as unsigned bigint) as tempusedMB,
        tu*100/tt as tempPerCent;
end
```

To display the output of `sp_iqspaceused`, run the procedure `myspace`:

```sql
myspace
```

**sp_iqstatistics procedure**

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns serial number, name, description, value, and unit specifier for each available statistic, or a specified statistic.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>sp_iqstatistics [ stat_name ]</code></td>
</tr>
<tr>
<td>Parameters</td>
<td><code>stat_name</code> (Optional) VARCHAR parameter specifying the name of a statistic.</td>
</tr>
<tr>
<td>Permissions</td>
<td>DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.</td>
</tr>
</tbody>
</table>

Reference: Building Blocks, Tables, and Procedures 461
System stored procedures

Description
When stat_name is provided, sp_iqstatistics returns one row for the given statistic, or zero rows if the name is invalid. When invoked without any parameter, sp_iqstatistics returns all statistics.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat_num</td>
<td>UNSIGNED INTEGER</td>
<td>Serial number of a statistic</td>
</tr>
<tr>
<td>stat_name</td>
<td>VARCHAR(255)</td>
<td>Name of statistic</td>
</tr>
<tr>
<td>stat_desc</td>
<td>VARCHAR(255)</td>
<td>Description of statistic</td>
</tr>
<tr>
<td>stat_value</td>
<td>LONG VARCHAR</td>
<td>Value of statistic</td>
</tr>
<tr>
<td>stat_unit</td>
<td>VARCHAR(128)</td>
<td>Unit specifier</td>
</tr>
</tbody>
</table>

The following statistics may be returned:

<table>
<thead>
<tr>
<th>stat_num</th>
<th>stat_name</th>
<th>stat_desc</th>
<th>stat_unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CpuTotalTime</td>
<td>Total CPU time in seconds consumed by the IQ server since last server startup</td>
<td>Second</td>
</tr>
<tr>
<td>1</td>
<td>CpuUserTime</td>
<td>CPU user time in seconds consumed by the IQ server since last server startup</td>
<td>Second</td>
</tr>
<tr>
<td>2</td>
<td>CpuSystemTime</td>
<td>CPU system time in seconds consumed by the IQ server since last server startup</td>
<td>Second</td>
</tr>
<tr>
<td>3</td>
<td>ThreadsFree</td>
<td>Number of IQ threads free</td>
<td>N/A</td>
</tr>
<tr>
<td>4</td>
<td>ThreadsInUse</td>
<td>Number of IQ threads in use</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>MemoryAllocated</td>
<td>Allocated memory in megabytes</td>
<td>MB</td>
</tr>
<tr>
<td>6</td>
<td>MemoryMaxAllocated</td>
<td>Max allocated memory in megabytes</td>
<td>MB</td>
</tr>
<tr>
<td>7</td>
<td>MainCacheCurrentSize</td>
<td>Main cache current size in megabytes</td>
<td>MB</td>
</tr>
<tr>
<td>8</td>
<td>MainCacheFinds</td>
<td>Main cache total number of lookup requests</td>
<td>N/A</td>
</tr>
<tr>
<td>9</td>
<td>MainCacheHits</td>
<td>Main cache total number of hits</td>
<td>N/A</td>
</tr>
<tr>
<td>10</td>
<td>MainCachePagesPinned</td>
<td>Main cache number of pages pinned</td>
<td>Page</td>
</tr>
<tr>
<td>11</td>
<td>MainCachePagesPinnedPercentage</td>
<td>Percentage of main cache pages pinned</td>
<td>%</td>
</tr>
<tr>
<td>12</td>
<td>MainCachePagesDirtyPercentage</td>
<td>Percentage of main cache pages dirted</td>
<td>%</td>
</tr>
<tr>
<td>13</td>
<td>MainCachePagesInUsePercentage</td>
<td>Percentage of main cache pages in use</td>
<td>%</td>
</tr>
<tr>
<td>14</td>
<td>TempCacheCurrentSize</td>
<td>Temporary cache current size in megabytes</td>
<td>MB</td>
</tr>
<tr>
<td>15</td>
<td>TempCacheFinds</td>
<td>Temporary cache total number of lookup requests</td>
<td>N/A</td>
</tr>
<tr>
<td>16</td>
<td>TempCacheHits</td>
<td>Temporary cache total number of hits</td>
<td>N/A</td>
</tr>
<tr>
<td>17</td>
<td>TempCachePagesPinned</td>
<td>Temporary cache number of pages pinned</td>
<td>Page</td>
</tr>
</tbody>
</table>
Example 1  Displays a single statistic, the total CPU time:

\[ \text{sp_iqstatistics 'CPUTotalTime'} \]

Example 2  Displays all statistics for \text{MainCache\%}:

\[ \text{SELECT * from sp_iqstatistics()} \text{ WHERE } \text{stat_name LIKE 'MainCache\%'} \]

\section*{sp_iqstatus procedure}

\textbf{Function}  Displays a variety of Sybase IQ status information about the current database.

\textbf{Syntax}  \texttt{sp_iqstatus}

\textbf{Permissions}  DBA authority required. Users without DBA authority must be granted \texttt{EXECUTE} permission to run the stored procedure.

\textbf{Description}  Shows status information about the current database, including the database name, creation date, page size, number of dbspaces, blocks usage, buffer usage, I/O, backup information, and so on.
sp_iqstatus displays an out-of-space status for main and temporary stores. If a store runs into an out-of-space condition, sp_iqstatus shows Y in the store’s out-of-space status display value.

sp_iqspaceused returns a subset of the same information as provided by sp_iqstatus, but allows the user to return the information in SQL variables to be used in calculations. See “sp_iqspaceused procedure” on page 460.

To display space that can be reclaimed by dropping connections, use sp_iqstatus and add the results from the two returned rows:

```sql
(First 2 rows)
```

The above example output shows that one active write transaction created 2175MB and destroyed 2850 MB of data. The total data consumed in transactions and not yet released is 4818MB, or 1968MB + 2850MB = 4818MB.

sp_iqstatus does not show blocks that will be deallocated at the next checkpoint. These blocks do however, appear in sp_iqdbspace output as type X.

The following output is from the sp_iqstatus stored procedure:
Create Time: 2009-04-03 11:30:20.674
Update Time: 2009-04-03 11:34:33.040
Main IQ Buffers: 255, 32Mb
Temporary IQ Buffers: 191, 24Mb
Main IQ Blocks Used: 5915 of 11200, 52%=46Mb, Max Block#:105278
Temporary IQ Blocks Used: 65 of 800, 8%=0Mb, Max Block#: 0
Main Reserved Blocks Available: 1600 of 1600, 100%=6Mb
Temporary Reserved Blocks Available: 6400 of 6400, 100%=50Mb
IQ Dynamic Memory: Current: 69mb, Max: 70mb
Main IQ Buffers: Used: 17, Locked: 0
Temporary IQ Buffers: Used: 4, Locked: 0
Main IQ I/O: I: L1581/P14 O: C3/D163/P161 D:34 C:97.1
Temporary IQ I/O: I: L6627/P0 O: C1086/D1166/P83 D:1082 C:100.0
Other Versions: 0 = 0Mb
Active Txn Versions: 0 = C:0Mb/D:0Mb
Last Full Backup ID: 0
Last Full Backup Time: None
Last Backup ID: 0
Last Backup Type: None
Last Backup Time: None
DB Updated: 1
Blocks in next ISF Backup: 0 Blocks: =0Mb
Blocks in next ISI Backup: 0 Blocks: =0Mb
DB File Encryption Status: OFF

The following is a key to understanding the Main IQ I/O and Temporary IQ I/O output codes:

- **I:** Input
- **L:** Logical pages read (“Finds”)
- **P:** Physical pages read
- **O:** Output
- **C:** Pages created
- **D:** Pages dirtied
- **P:** Physically written
- **D:** Pages destroyed
- **C:** Compression ratio

See also sp_iqtransaction procedure and sp_iqversionuse procedure
System stored procedures

**sp_iqsysmon procedure**

**Function**
Monitors multiple components of Sybase IQ, including the management of buffer cache, memory, threads, locks, I/O functions, and CPU utilization.

**Batch mode syntax**
- `sp_iqsysmon start_monitor`
- `sp_iqsysmon stop_monitor [ , "section(s)" ]`
- `sp_iqsysmon "time-period" [ , "section(s)" ]`

**File mode syntax**
- `sp_iqsysmon start_monitor, "filemode" [ , "monitor-options" ]`
- `sp_iqsysmon stop_monitor`

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Batch mode usage**
- **start_monitor** Starts monitoring.
- **stop_monitor** Stops monitoring and displays the report.
- **time-period** The time period for monitoring. Must be in the form HH:MM:SS.
- **section(s)** The abbreviation for one or more sections to be displayed by sp_iqsysmon. When more than one section is specified, the section abbreviations must be separated by spaces and the list must be enclosed in single or double quotes. The default is to display all sections.

For the sections related to IQ store, you can specify main or temporary store by prefixing the section abbreviation with “m” or “t”, respectively. See Table 7-56. Without the prefix, both stores are monitored. For example, if you specify “mbufman”, only the IQ main store buffer manager is monitored. If you specify “mbufman tbufman” or “bufman”, both the main and temporary store buffer managers are monitored.
Table 7-56: sp_iqsysmon report section abbreviations

<table>
<thead>
<tr>
<th>Report section or IQ component</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer manager</td>
<td>(m/t)bufman</td>
</tr>
<tr>
<td>Buffer pool</td>
<td>(m/t)bufpool</td>
</tr>
<tr>
<td>Prefetch management</td>
<td>(m/t)prefetch</td>
</tr>
<tr>
<td>Free list management</td>
<td>(m/t)freelist</td>
</tr>
<tr>
<td>Buffer allocation</td>
<td>(m/t)bufalloc</td>
</tr>
<tr>
<td>Memory management</td>
<td>memory</td>
</tr>
<tr>
<td>Thread management</td>
<td>threads</td>
</tr>
<tr>
<td>CPU utilization</td>
<td>cpu</td>
</tr>
<tr>
<td>Transaction management</td>
<td>txn</td>
</tr>
<tr>
<td>Server context statistics</td>
<td>server</td>
</tr>
<tr>
<td>Catalog statistics</td>
<td>catalog</td>
</tr>
</tbody>
</table>

Note The Sybase IQ components Disk I/O and lock manager are not currently supported by sp_iqsysmon.

File mode usage

- **start_monitor** Starts monitoring.
- **stop_monitor** Stops monitoring and writes the remaining output to the log file.
- **filemode** Specifies that sp_iqsysmon is running in file mode. In file mode, a sample of statistics is displayed for every interval in the monitoring period. By default, the output is written to a log file named dbname.connid-iqmon. Use the file_suffix option to change the suffix of the output file. See the monitor_options parameter for a description of the file_suffix option.
- **monitor_options** The monitor_options string can include one or more of the following options:
  - `-interval seconds`
    Specifies the reporting interval in seconds. A sample of monitor statistics is output to the log file after every interval. The default is every 60 seconds, if the -interval option is not specified. The minimum reporting interval is 2 seconds. If the interval specified for this option is invalid or less than 2 seconds, the interval is set to 2 seconds.
System stored procedures

The first display shows the counters from the start of the server. Subsequent displays show the difference from the previous display. You can usually obtain useful results by running the monitor at the default interval of 60 seconds during a query with performance problems or during a time of day with performance problems. A very short interval may not provide meaningful results. The interval should be proportional to the job time; 60 seconds is usually more than enough time.

- `-file_suffix suffix`
  Creates a monitor output file named `dbname.connid-suffix`. If you do not specify the `-file_suffix` option, the suffix defaults to `iqmon`. If you specify the `-file_suffix` option and do not provide a suffix or provide a blank string as a suffix, no suffix is used.

- `-append` or `-truncate`
  Directs `sp_iqsysmon` to append to the existing output file or truncate the existing output file, respectively. Truncate is the default. If both options are specified, the option specified later in the string is effective.

- `-section section(s)`
  Specifies the abbreviation of one or more sections to write to the monitor log file. The default is to write all sections. The abbreviations specified in the sections list in file mode are the same abbreviations used in batch mode. See Table 7-56 for a list of abbreviations. When more than one section is specified, spaces must separate the section abbreviations.

If the `-section` option is specified with no sections, none of the sections are monitored. An invalid section abbreviation is ignored and a warning is displayed in the IQ message file.
The sp_iqsysmon stored procedure monitors multiple components of Sybase IQ, including the management of buffer cache, memory, threads, locks, I/O functions, and CPU utilization.

The sp_iqsysmon procedure supports two modes of monitoring:

- **Batch mode**
  
  In batch mode, sp_iqsysmon collects the monitor statistics for the period between starting and stopping the monitor or for the time period specified in the *time-period* parameter. At the end of the monitoring period, sp_iqsysmon displays a list of consolidated statistics.

  sp_iqsysmon in batch mode is similar to the Adaptive Server Enterprise procedure sp_sysmon.

- **File mode**
  
  In file mode, sp_iqsysmon writes the sample statistics in a log file for every interval period between starting and stopping the monitor.

  Note that the first display in file mode shows the counters from the start of the server. Subsequent displays show the difference from the previous display.

  sp_iqsysmon in file mode is similar to the IQ UTILITIES command START MONITOR and STOP MONITOR interface.

### Table 7-57: sp_iqsysmon usage examples

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_iqsysmon start_monitor</td>
<td>Starts the monitor in batch mode and displays all sections for main and temporary store</td>
</tr>
<tr>
<td>sp_iqsysmon stop_monitor</td>
<td></td>
</tr>
<tr>
<td>sp_iqsysmon start_monitor “mbufman mbufpool”</td>
<td>Starts the monitor in batch mode and displays the Buffer Manager and Buffer Pool statistics for main store</td>
</tr>
<tr>
<td>sp_iqsysmon stop_monitor “mbufpool memory”</td>
<td>Runs the monitor in batch mode for 10 seconds and displays the consolidated statistics at the end of the time period</td>
</tr>
<tr>
<td>sp_iqsysmon start_monitor, 'filemode', ‘-interval 5 -sections mbufpool memory’</td>
<td>Starts the monitor in file mode and writes to the log file every 5 seconds the statistics for Main Buffer Pool and Memory Manager</td>
</tr>
<tr>
<td>sp_iqsysmon stop_monitor</td>
<td></td>
</tr>
</tbody>
</table>

**Batch mode examples**

Prints monitor information after 10 minutes:

```
sp_iqsysmon "00:10:00"
```
Prints only the Memory Manager section of the `sp_iqsysmon` report after 5 minutes:

```
sp_iqsysmon "00:05:00", memory
```

Starts the monitor, executes two procedures and a query, stops the monitor, then prints only the Buffer Manager section of the report:

```
sp_iqsysmon start_monitor
go
execute proc1
go
execute proc2
go
select sum(total_sales) from titles
go
sp_iqsysmon stop_monitor, bufman
go
```

Prints only the Main Buffer Manager and Main Buffer Pool sections of the report after 20 minutes:

```
sp_iqsysmon "00:02:00", "mbufman mbufpool"
```

### File mode examples

Truncates and writes information to the log file every 2 seconds between starting the monitor and stopping the monitor:

```
sp_iqsysmon start_monitor, 'filemode', '-interval 2'
.
.
sp_iqsysmon stop_monitor
```

Appends output for only the Main Buffer Manager and Memory Manager sections to an ASCII file with the name `dbname.connid-testmon`. For the database `iqdemo`, writes results in the file `iqdemo.2-testmon`:

```
sp_iqsysmon start_monitor, 'filemode', 
"-file_suffix testmon -append -section mbufman memory"
.
.
sp_iqsysmon stop_monitor
```

### Example

Run the monitor in batch mode for 10 seconds and display the consolidated statistics at the end of the time period

```
sp_iqsysmon "00:00:10", "mbufpool memory"
```

======================================
### Buffer Pool (Main)

```
<table>
<thead>
<tr>
<th>STATS-NAME</th>
<th>TOTAL</th>
<th>NONE</th>
<th>BTREEV</th>
<th>BTREEF</th>
<th>BV</th>
<th>VDO</th>
<th>DBEXT</th>
<th>DBID</th>
<th>SORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovedToMRU</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MovedToWash</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RemovedFromLRU</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RemovedFromWash</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RemovedInScanMode</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STORE</th>
<th>GARRAY</th>
<th>BARRAY</th>
<th>BLKMAP</th>
<th>HASH</th>
<th>CKPT</th>
<th>BM</th>
<th>TEST</th>
<th>CMID</th>
<th>RIDCA</th>
<th>LOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STATS-NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>127 (100.0 %)</td>
</tr>
<tr>
<td>InUse</td>
<td>4 (3.1 %)</td>
</tr>
<tr>
<td>Dirty</td>
<td>1 (0.8 %)</td>
</tr>
<tr>
<td>Pinned</td>
<td>0 (0.0 %)</td>
</tr>
<tr>
<td>Flushes</td>
<td>0</td>
</tr>
<tr>
<td>FlushedBufferCount</td>
<td>0</td>
</tr>
<tr>
<td>GetPageFrame</td>
<td>0</td>
</tr>
<tr>
<td>GetPageFrameFailure</td>
<td>0</td>
</tr>
<tr>
<td>GotEmptyFrame</td>
<td>0</td>
</tr>
<tr>
<td>Washed</td>
<td>0</td>
</tr>
<tr>
<td>TimesSweepersWoken</td>
<td>0</td>
</tr>
<tr>
<td>washTeamSize</td>
<td>0</td>
</tr>
<tr>
<td>WashMaxSize</td>
<td>26 (20.5 %)</td>
</tr>
<tr>
<td>washNBuffers</td>
<td>4 (3.1 %)</td>
</tr>
<tr>
<td>washNDirtyBuffers</td>
<td>1 (0.8 %)</td>
</tr>
<tr>
<td>washSignalThreshold</td>
<td>3 (2.4 %)</td>
</tr>
<tr>
<td>washNActiveSweepers</td>
<td>0</td>
</tr>
<tr>
<td>washIntensity</td>
<td>1</td>
</tr>
</tbody>
</table>
```

### Memory Manager

```
<table>
<thead>
<tr>
<th>STATS-NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemAllocated</td>
<td>43616536 (42594 KB)</td>
</tr>
<tr>
<td>MemAllocatedMax</td>
<td>43735080 (42710 KB)</td>
</tr>
<tr>
<td>MemAllocatedEver</td>
<td>0 (0 KB)</td>
</tr>
<tr>
<td>MemNAllocated</td>
<td>67079</td>
</tr>
<tr>
<td>MemNAllocatedEver</td>
<td>0</td>
</tr>
</tbody>
</table>
```
System stored procedures

<table>
<thead>
<tr>
<th>MemNTimesLocked</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemNTimesWaited</td>
<td>0 ( 0.0 %)</td>
</tr>
</tbody>
</table>

See also
IQ UTILITIES statement in Chapter 1, “SQL Statements,” in *Reference: Statements and Options*

Chapter 5, “Monitoring and Tuning Performance” in the *Performance and Tuning Guide*

**sp_iqtable procedure**

**Function**
Displays information about tables in the database.

**Syntax1**
```
sp_iqtable ([table_name], [table_owner], [table_type])
```

**Syntax2**
```
sp_iqtable [table_name=’tablename’],
[table_owner=’tableowner’], [table_type=’tabletype’]
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**
**Syntax1**
If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example, `sp_iqtable NULL,NULL,TEMP` and `sp_iqtable NULL,dbo,SYSTEM`.

**Note**
The `table_type` values ALL and VIEW must be enclosed in single quotes in Syntax1.

**Syntax2**
The parameters can be specified in any order. Enclose them in single quotes.

Table 7-58 lists the allowed values for the `table_type` parameter:
### Table 7-58: sp_iqtable table_type values

<table>
<thead>
<tr>
<th>table_type value</th>
<th>Information displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>System tables</td>
</tr>
<tr>
<td>TEMP</td>
<td>Global temporary tables</td>
</tr>
<tr>
<td>VIEW</td>
<td>Views</td>
</tr>
<tr>
<td>ALL</td>
<td>IQ tables, system tables, and views</td>
</tr>
<tr>
<td>any other value</td>
<td>IQ tables</td>
</tr>
</tbody>
</table>

**Description**

Specifying one parameter returns only the tables that match that parameter. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all Sybase IQ tables in the database. There is no method for returning the names of local temporary tables.
### Table 7-59: sp_iqtable columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>table_type</td>
<td>BASE – a base table. MAT VIEW - a materialized view. GBL TEMP - a global temporary table. PARTITION - a table partition (this table is for internal use only and cannot be used by Sybase IQ users). VIEW – a view. JVT – a join virtual table.</td>
</tr>
<tr>
<td>table_owner</td>
<td>The owner of the table</td>
</tr>
<tr>
<td>server_type</td>
<td>IQ – an object created in the IQ store. SA – an object created in the SA store. All views are created in the SA store.</td>
</tr>
<tr>
<td>location</td>
<td>TEMP - IQ temporary store. MAIN – IQ store. SYSTEM – catalog store.</td>
</tr>
<tr>
<td>dbspace_id</td>
<td>Name of the dbspace where the table resides.</td>
</tr>
<tr>
<td>isPartitioned</td>
<td>'Y' if the column belongs to a partitioned table and has one or more partitions whose dbspace is different from the table partition’s dbspace, 'N' if the column’s table is not partitioned or each partition of the column resides in the same dbspace as the table partition.</td>
</tr>
<tr>
<td>remarks</td>
<td>User comments added with the COMMENT statement.</td>
</tr>
<tr>
<td>table_constraints</td>
<td>Constraints against the table.</td>
</tr>
</tbody>
</table>

#### Examples

The following variations in syntax both return information about the table Departments:

```
sp_iqtable ('Departments')
sp_iqtable table_name='Departments'
```

<table>
<thead>
<tr>
<th>Table_name</th>
<th>Table_type</th>
<th>Table_owner</th>
<th>Server_type</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Departments</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
<tr>
<td>dbspace_id</td>
<td>16387</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following variations in syntax both return all tables that are owned by table owner GROUPO:

```
sp_iqtable ('Departments')
sp_iqtable table_name= 'Departments'
```
sp_iqtable table_owner='GROUPO'

**Table**

<table>
<thead>
<tr>
<th>Table_name</th>
<th>Table_type</th>
<th>Table_owner</th>
<th>Server_type</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contacts</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
<tr>
<td>Customers</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
<tr>
<td>Departments</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
<tr>
<td>Employees</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
<tr>
<td>FinancialCodes</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
<tr>
<td>FinancialData</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
<tr>
<td>Products</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
<tr>
<td>SalesOrders</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
<tr>
<td>SalesOrderItems</td>
<td>BASE</td>
<td>GROUPO</td>
<td>IQ</td>
<td>Main</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dbspace_id</th>
<th>isPartitioned</th>
<th>Remarks</th>
<th>table_constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>16387</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
<tr>
<td>16387</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
<tr>
<td>16387</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
<tr>
<td>16387</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
<tr>
<td>16387</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
<tr>
<td>16387</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
<tr>
<td>16387</td>
<td>N</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

**sp_iqtablesize procedure**

**Function**

Returns the size of the specified table.

**Syntax**

```
sp_iqtablesize ( table_owner.table_name )
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**

Returns the total size of the table in KBytes and NBBlocks (IQ blocks). Also returns the number of pages required to hold the table in memory, and the number of IQ pages that are compressed when the table is compressed (on disk). You must specify the `table_name` parameter with this procedure. If you are the owner of `table_name`, then you do not have to specify the `table_owner` parameter.
Table 7-60: sp_iqtablesize columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ownername</td>
<td>Name of owner</td>
</tr>
<tr>
<td>Tablename</td>
<td>Name of table</td>
</tr>
<tr>
<td>Columns</td>
<td>Number of columns in the table</td>
</tr>
<tr>
<td>KBytes</td>
<td>Physical table size in KB</td>
</tr>
<tr>
<td>Pages</td>
<td>Number of IQ pages needed to hold the table in memory</td>
</tr>
<tr>
<td>CompressedPages</td>
<td>Number of IQ pages that are compressed, when the table is compressed (on disk)</td>
</tr>
<tr>
<td>NBloksk</td>
<td>Number of IQ blocks</td>
</tr>
</tbody>
</table>

Pages is the total number of IQ pages for the table. The unit of measurement for pages is IQ page size. All in-memory buffers (buffers in the IQ buffer cache) are the same size.

IQ pages on disk are compressed. Each IQ page on disk uses 1 to 16 blocks. If the IQ page size is 128KB, then the IQ block size is 8KB. In this case, an individual on-disk page could be 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, or 128 KB.

If you divide the KBytes value by page size, you see the average on-disk page size.

Note Sybase IQ always reads and writes an entire page, not blocks. For example, if an individual page compresses to 88K, then IQ reads and writes the 88K in one I/O. The average page is compressed by a factor of 2 to 3.

NBloksk is Kbytes divided by IQ block size.

CompressedPages is the number of pages that are compressed. For example, if Pages is 1000 and CompressedPages is 992, this means that 992 of the 1000 pages are compressed. CompressedPages divided by Pages is usually near 100%, because most pages compress. An empty page is not compressed, since Sybase IQ does not write empty pages. IQ pages compress well, regardless of the fullness of the page.

Example
call sp_iqtablesize ('dba.emp1')

<table>
<thead>
<tr>
<th>Ownername</th>
<th>Tablename</th>
<th>Columns</th>
<th>KBytes</th>
<th>Pages</th>
<th>CompressedPages</th>
<th>NBloksk</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA</td>
<td>empl</td>
<td>4</td>
<td>1504</td>
<td>24</td>
<td>14</td>
<td>188</td>
</tr>
</tbody>
</table>
**sp_iqtableuse procedure**

**Function**
Reports detailed usage information for tables accessed by the workload.

**Syntax**
```
sp_iqtableuse
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Tables created in SYSTEM are not reported.

**Table 7-61: sp_iqtableuse columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>Table name</td>
</tr>
<tr>
<td>Owner</td>
<td>Username of table owner</td>
</tr>
<tr>
<td>UID**</td>
<td>Table unique identifier</td>
</tr>
<tr>
<td>LastDT</td>
<td>Date/time of last access</td>
</tr>
<tr>
<td>NRef</td>
<td>Number of query references</td>
</tr>
</tbody>
</table>

**UID** is a number assigned by the system that uniquely identifies the instance of the table (where instance is defined when an object is created).

**See also**


“INDEX_ADVISOR option” in Chapter 2, “Database Options,” in Reference: Statements and Options

**sp_iqtransaction procedure**

**Function**
Shows information about transactions and versions.

**Syntax**
```
sp_iqtransaction
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.


**System stored procedures**

**Description**

`sp_iqtransaction` returns a row for each transaction control block in the Sybase IQ transaction manager. The columns Name, Userid, and ConnHandle are the connection properties Name, Userid, and Number, respectively. Rows are ordered by TxnID.

`sp_iqtransaction` output does not contain rows for connections that do not have a transaction started. To see all connections, use `sp_iqconnection`.

**Note**

Although you can use `sp_iqtransaction` to identify users who are blocking other users from writing to a table, `sp_iqlocks` is a better choice for this purpose.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the application.</td>
</tr>
<tr>
<td>Userid</td>
<td>The user ID for the connection.</td>
</tr>
<tr>
<td>TxnID</td>
<td>The transaction ID of this transaction control block. The transaction ID is assigned during begin transaction. This is the same as the transaction ID displayed in the <code>.iqmsg</code> file by the BeginTxn, CmtTxn and PostCmtTxn messages as well as the Txn ID Seq logged when the database is opened.</td>
</tr>
<tr>
<td>CmtID</td>
<td>The ID assigned by the transaction manager when the transaction commits. It is zero for active transactions.</td>
</tr>
<tr>
<td>VersionID</td>
<td>In simplex databases, the VersionID is the same as the TxnID. For the multiplex coordinator, the VersionID is the same as the TxnID of the active transaction and VersionID is the same as the CmtID of a committed transaction. In multiplex secondary servers, the VersionID is the CmtID of the transaction that created the database version on the multiplex coordinator. It is used internally by the Sybase IQ in-memory catalog and the IQ transaction manager to uniquely identify a database version to all nodes within a multiplex database.</td>
</tr>
<tr>
<td>State</td>
<td>The state of the transaction control block. This variable reflects internal Sybase IQ implementation detail and is subject to change in the future. At the time of this writing, transaction states are NONE, ACTIVE, ROLLING_BACK, ROLLED_BACK, COMMITTING, COMMITTED, and APPLIED.</td>
</tr>
<tr>
<td>ConnHandle</td>
<td>The ID number of the connection.</td>
</tr>
<tr>
<td>IQConnID</td>
<td>The ten-digit connection ID displayed as part of all messages in the <code>.iqmsg</code> file. This is a monotonically increasing integer unique within a server session.</td>
</tr>
<tr>
<td>MainTableKBCr</td>
<td>The number of kilobytes of IQ store space created by this transaction.</td>
</tr>
<tr>
<td>MainTableKBDr</td>
<td>The number of kilobytes of IQ store space dropped by this transaction, but which persist on disk in the store because the space is visible in other database versions or other savepoints of this transaction.</td>
</tr>
<tr>
<td>TempTableKBCr</td>
<td>The number of kilobytes of IQ temporary store space created by this transaction for storage of IQ temporary table data.</td>
</tr>
</tbody>
</table>
### Example

Here is an example of `sp_iqtransaction` output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Userid</th>
<th>TxnID</th>
<th>CmtID</th>
<th>VersionID</th>
<th>State</th>
<th>ConnHandle</th>
<th>IQConnID</th>
</tr>
</thead>
<tbody>
<tr>
<td>red2</td>
<td>DBA</td>
<td>10058</td>
<td>10700</td>
<td>10058</td>
<td>COMMITTED</td>
<td>419740283</td>
<td>14</td>
</tr>
<tr>
<td>blue1</td>
<td>DBA</td>
<td>10568</td>
<td>0</td>
<td>10568</td>
<td>ACTIVE</td>
<td>640038605</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>DBA</td>
<td>10604</td>
<td>0</td>
<td>10604</td>
<td>ACTIVE</td>
<td>2094200996</td>
<td>18</td>
</tr>
<tr>
<td>fromSCJ</td>
<td>DBA</td>
<td>10619</td>
<td>0</td>
<td>10619</td>
<td>ACTIVE</td>
<td>954498130</td>
<td>20</td>
</tr>
<tr>
<td>blue2</td>
<td>DBA</td>
<td>10634</td>
<td>10677</td>
<td>10634</td>
<td>COMMITTED</td>
<td>167015670</td>
<td>21</td>
</tr>
<tr>
<td>ntJava2</td>
<td>DBA</td>
<td>10676</td>
<td>0</td>
<td>10676</td>
<td>ACTIVE</td>
<td>1779741471</td>
<td>24</td>
</tr>
<tr>
<td>blue2</td>
<td>DBA</td>
<td>10678</td>
<td>0</td>
<td>10678</td>
<td>ACTIVE</td>
<td>167015670</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TempTableKBDr</td>
<td>The number of kilobytes of IQ temporary table space dropped by this transaction, but which persist on disk in the IQ temporary store because the space is visible to IQ cursors or is owned by other savepoints of this transaction.</td>
</tr>
<tr>
<td>TempWorkSpaceKB</td>
<td>For ACTIVE transactions, this is a snapshot of the work space in use at this instant by this transaction, such as sorts, hashes, and temporary bitmaps. The number varies depending on when you run <code>sp_iqtransaction</code>. For example, the query engine might create 60MB in the temporary cache but release most of it quickly, even though query processing continues. If you run <code>sp_iqtransaction</code> after the query finishes, this column shows a much smaller number. When the transaction is no longer active, this column is zero. For ACTIVE transactions, this column is the same as the TempWorkSpaceKB column of <code>sp_iqconnection</code>.</td>
</tr>
<tr>
<td>TxnCreateTime</td>
<td>The time the transaction began. All Sybase IQ transactions begin implicitly as soon as an active connection is established or when the previous transaction commits or rolls back.</td>
</tr>
<tr>
<td>CursorCount</td>
<td>The number of open Sybase IQ cursors that reference this transaction control block. If the transaction is ACTIVE, it indicates the number of open cursors created within the transaction. If the transaction is COMMITTED, it indicates the number of HOLD cursors that reference a database version owned by this transaction control block.</td>
</tr>
<tr>
<td>SpCount</td>
<td>The number of savepoint structures that exist within the transaction control block. Savepoints may be created and released implicitly. Therefore, this number does not indicate the number of user-created savepoints within the transaction.</td>
</tr>
<tr>
<td>SpNumber</td>
<td>The active savepoint number of the transaction. This is an implementation detail and might not reflect a user-created savepoint.</td>
</tr>
<tr>
<td>MPXServerName</td>
<td>The value indicates if an active transaction is from an inter-node communication (INC) connection. If from INC connection, the value is the name of the multiplex server where the transaction originates. NULL if not from an INC connection. Always NULL if the transaction is not active.</td>
</tr>
<tr>
<td>GlobalTxnID</td>
<td>The value indicates the global transaction ID associated with the current transaction. Zero if there is no associated global transaction.</td>
</tr>
</tbody>
</table>
## System stored procedures

<table>
<thead>
<tr>
<th>DBA</th>
<th>GlobalTxnID</th>
<th>ACTIVE</th>
<th>PID</th>
<th>Start</th>
<th>TxnCreateTime</th>
<th>CursorCount</th>
<th>SpCount</th>
<th>SpNumber</th>
<th>TempWorkSpaceKB</th>
<th>MPXServerName</th>
<th>GlobalTxnID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntl</td>
<td>10699</td>
<td>ACTIVE</td>
<td>0</td>
<td>10699</td>
<td>2009-06-26</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>102592</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>red2</td>
<td>10701</td>
<td>ACTIVE</td>
<td>0</td>
<td>10701</td>
<td>2009-06-26</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>419740283</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16687</td>
<td>ACTIVE</td>
<td>0</td>
<td>16687</td>
<td>2009-06-26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1306718536</td>
<td>(NULL)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TempTableKBCr</th>
<th>TempTableKBDr</th>
<th>MainTableKBCr</th>
<th>MainTableKBDr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>65824</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2912</td>
<td>22096</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TempWorkSpaceKB</th>
<th>TxnCreateTime</th>
<th>CursorCount</th>
<th>SpCount</th>
<th>SpNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2009-06-26</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>102592</td>
<td>2009-06-26</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2009-06-26</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2009-06-26</td>
<td>0</td>
<td>24</td>
<td>262</td>
</tr>
<tr>
<td>0</td>
<td>2009-06-26</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2009-06-26</td>
<td>0</td>
<td>39</td>
<td>408</td>
</tr>
<tr>
<td>128</td>
<td>2009-06-26</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2009-06-26</td>
<td>0</td>
<td>42</td>
<td>413</td>
</tr>
<tr>
<td>680</td>
<td>2009-06-26</td>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>0</td>
<td>2009-06-26</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SP Number</th>
<th>SP Count</th>
<th>Cursor Count</th>
<th>TempWorkSpaceKB</th>
<th>TxnCreateTime</th>
<th>MPXServerName</th>
<th>GlobalTxnID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>680</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>102592</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2009-06-26</td>
<td>(NULL)</td>
<td></td>
</tr>
</tbody>
</table>
**sp_iqunusedcolumn procedure**

**Function**
Reports IQ columns that were not referenced by the workload.

**Syntax**

```
sp_iqunusedcolumn
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Columns from tables created in SYSTEM or local temporary tables are not reported.

**Table 7-63: sp_iqunusedcolumn columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>Table name</td>
</tr>
<tr>
<td>ColumnName</td>
<td>Column name</td>
</tr>
<tr>
<td>Owner</td>
<td>Username of column owner</td>
</tr>
</tbody>
</table>

**Example**
Sample output from the `sp_iqunusedcolumn` procedure:

<table>
<thead>
<tr>
<th>TableName</th>
<th>ColumnName</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>SalesOrders</td>
<td>ID</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrders</td>
<td>CustomerID</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrders</td>
<td>OrderDate</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrders</td>
<td>FinancialCode</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrders</td>
<td>Region</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrders</td>
<td>SalesRepresentative</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrderItems</td>
<td>ID</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrderItems</td>
<td>LineID</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrderItems</td>
<td>ProductID</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrderItems</td>
<td>Quantity</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrderItems</td>
<td>ShipDate</td>
<td>GROUPO</td>
</tr>
<tr>
<td>Contacts</td>
<td>ID</td>
<td>GROUPO</td>
</tr>
<tr>
<td>Contacts</td>
<td>Surname</td>
<td>GROUPO</td>
</tr>
<tr>
<td>Contacts</td>
<td>GivenName</td>
<td>GROUPO</td>
</tr>
</tbody>
</table>

**See also**

“INDEX_ADVISOR option” in Chapter 2, “Database Options,” in Reference: Statements and Options

sp_iqunusedindex procedure

Function          Reports IQ secondary (non-FP) indexes that were not referenced by the workload.

Syntax           \texttt{sp_iqunusedindex}

Permissions       DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description       Indexes from tables created in SYSTEM or local temporary tables are not reported.

\textit{Table 7-64: sp_iqunusedindex columns}

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndexName</td>
<td>Index name</td>
</tr>
<tr>
<td>TableName</td>
<td>Table name</td>
</tr>
<tr>
<td>Owner</td>
<td>User name of index owner</td>
</tr>
<tr>
<td>IndexType</td>
<td>Index type</td>
</tr>
</tbody>
</table>

Example          Sample output from the \texttt{sp_iqunusedindex} procedure:
### sp_iqunusedtable procedure

**Function**: Reports IQ tables that were not referenced by the workload.

**Syntax**

```sql
sp_iqunusedtable
```

**Permissions**: DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**: Tables created in SYSTEM and local temporary tables are not reported.

---

<table>
<thead>
<tr>
<th>IndexName</th>
<th>TableName</th>
<th>Owner</th>
<th>IndexType</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIQ_IDX_T450_I7_HG</td>
<td>SalesOrders</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T450_C6_HG</td>
<td>SalesOrders</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T450_C4_HG</td>
<td>SalesOrders</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T450_C2_HG</td>
<td>SalesOrders</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T451_I6_HG</td>
<td>SalesOrderItems</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T451_C3_HG</td>
<td>SalesOrderItems</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T451_C1_HG</td>
<td>SalesOrderItems</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T452_I11_HG</td>
<td>Contacts</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T453_I10_HG</td>
<td>Contacts</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T454_I4_HG</td>
<td>FinancialCodes</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T455_I5_HG</td>
<td>FinancialData</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T455_C3_HG</td>
<td>FinancialData</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T456_I8_HG</td>
<td>Products</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T457_I4_HG</td>
<td>Departments</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T457_C3_HG</td>
<td>Departments</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T458_I21_HG</td>
<td>Departments</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
<tr>
<td>ASIQ_IDX_T458_C5_HG</td>
<td>Departments</td>
<td>GROUPO</td>
<td>HG</td>
</tr>
</tbody>
</table>

See also


“INDEX_ADVISOR option” in Chapter 2, “Database Options,” in *Reference: Statements and Options*
**Table 7-65: sp_iqunusedtable columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>Table name</td>
</tr>
<tr>
<td>Owner</td>
<td>Username of table owner</td>
</tr>
</tbody>
</table>

**Example**

The following table illustrates sample output from the `sp_iqunusedtable` procedure.

<table>
<thead>
<tr>
<th>TableName</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>FinancialCodes</td>
<td>GROUPO</td>
</tr>
<tr>
<td>Contacts</td>
<td>GROUPO</td>
</tr>
<tr>
<td>Employees</td>
<td>GROUPO</td>
</tr>
<tr>
<td>empl</td>
<td>DBA</td>
</tr>
<tr>
<td>SalesOrders</td>
<td>GROUPO</td>
</tr>
<tr>
<td>FinancialData</td>
<td>GROUPO</td>
</tr>
<tr>
<td>Departments</td>
<td>GROUPO</td>
</tr>
<tr>
<td>SalesOrderItems</td>
<td>GROUPO</td>
</tr>
<tr>
<td>Products</td>
<td>GROUP</td>
</tr>
<tr>
<td>iq_dummy</td>
<td>DBA</td>
</tr>
<tr>
<td>Customers</td>
<td>GROUPO</td>
</tr>
<tr>
<td>sale</td>
<td>DBA</td>
</tr>
</tbody>
</table>

**See also**


**sp_iqversionuse procedure**

**Function**

Displays version usage for the IQ main store.

**Syntax**

```
sp_iqversionuse
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.
Description

The `sp_iqversionuse` system stored procedure helps troubleshoot situations where the databases uses excessive storage space due to multiple table versions.

If out-of-space conditions occur or `sp_iqstatus` shows a high percentage of main blocks in use on a multiplex server, run `sp_iqversionuse` to find out which versions are being used and the amount of space that can be recovered by releasing versions. For information on multiplex capability, see *Using Sybase IQ Multiplex*.

The amount of space is expressed as a range because the actual amount typically depends on which other versions are released. The actual amount of space released can be anywhere between the values of `MinKBRelease` and `MaxKBRelease`. The oldest version always has `MinKBRelease` equal to `MaxKBRelease`.

`WasReported` indicates whether version usage information has been sent from the secondary server to the coordinator. `WasReported` is 0 initially on a coordinator for new versions. `WasReported` changes to 1 once SQL Remote replicates version usage information back to the coordinator. If `WasReported` is 0 for an extended period, SQL Remote might be stopped.

**Note** The `WasReported` column is used in a multiplex setting. For more information on multiplex, see *Using Sybase IQ Multiplex*.

### Table 7-66: `sp_iqversionuse` columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VersionID</td>
<td>The version identifier</td>
</tr>
<tr>
<td>Server</td>
<td>The server to which users of this version are connected</td>
</tr>
<tr>
<td>IQConnID</td>
<td>The connection ID using this version</td>
</tr>
<tr>
<td>WasReported</td>
<td>Indicates whether the server has received usage information for this version</td>
</tr>
<tr>
<td>MinKBRelease</td>
<td>The minimum amount of space returned once this version is no longer in use</td>
</tr>
<tr>
<td>MaxKBRelease</td>
<td>The maximum amount of space returned once this version is no longer in use</td>
</tr>
</tbody>
</table>

**Example**

The following table illustrates sample output from the `sp_iqversionuse` system procedure:

```
<table>
<thead>
<tr>
<th>VersionID</th>
<th>Server</th>
<th>IQConnID</th>
<th>WasReported</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ab2ab_iqdemo</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>
```
System stored procedures

<table>
<thead>
<tr>
<th>MinKBRelease</th>
<th>MaxKBRelease</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

sp_iqview procedure

**Function**
Displays information about views in a database.

**Syntax1**
```
sp_iqview ([view_name],[view_owner],[view_type])
```

**Syntax2**
```
sp_iqview [view_name='viewname'],
[view_owner='viewowner'],[view_type='viewtype']
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**

**Syntax1**
```
sp_iqview NULL,NULL,SYSTEM
```

If you do not specify either of the first two parameters, but specify the next parameter in the sequence, you must substitute NULL for the omitted parameters. For example:
```
sp_iqview
NULL,NULL,SYSTEM
```
```
and
```
sp_iqview deptview,NULL,'ALL'
```

**Note** The `view_type` value ALL must be enclosed in single quotes in Syntax1.

**Syntax2**
The parameters can be specified in any order. Enclose them in single quotes.

Table 7-67 lists the allowed values for the `view_type` parameter.

<table>
<thead>
<tr>
<th>view_type value</th>
<th>Information displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>System views</td>
</tr>
<tr>
<td>ALL</td>
<td>User and system views</td>
</tr>
<tr>
<td>any other value</td>
<td>User views</td>
</tr>
</tbody>
</table>

**Description**
Specifying one of the parameters returns only the views with the specified view name or views that are owned by the specified user. Specifying more than one parameter filters the results by all of the parameters specified. Specifying no parameters returns all user views in a database.
Table 7-68: `sp_iqview` columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>view_name</td>
<td>The name of the view</td>
</tr>
<tr>
<td>view_owner</td>
<td>The owner of the view</td>
</tr>
<tr>
<td>view_def</td>
<td>The view definition as specified in the CREATE VIEW statement</td>
</tr>
<tr>
<td>remarks</td>
<td>User comments added with the COMMENT statement</td>
</tr>
</tbody>
</table>

`sp_iqview` returns a view definition greater than 32K characters without truncation.

Examples

The following variations in syntax both return information about the view `deptview`:

```sql
call sp_iqview('ViewSalesOrders')
sp_iqview view_name='ViewSalesOrders'
```

The following variations in syntax both return all views that are owned by view owner `GROUPO`:

```sql
sp_iqview NULL,GROUPO
sp_iqview view_owner='GROUPO'
```

<table>
<thead>
<tr>
<th>view_name</th>
<th>view_owner</th>
<th>view_def</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViewSalesOrders</td>
<td>GROUPO</td>
<td>Create views GROUPO, ViewSalesOrders( ID, LineID, ProductID, Quantity, OrderDate, ShipDate, Region, SalesRepresentativeName</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

**sp_iqwho procedure**

Function

Displays information about all current users and connections, or about a particular user or connection.

Syntax

```sql
sp_iqwho [{ connhandle | user-name } [ , arg-type ] ]
```

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

The `sp_iqwho` stored procedure displays information about all current users and connections, or about a particular user or connection.
### Table 7-69: sp_iqwho columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnHandle</td>
<td>The SA connection handle</td>
</tr>
<tr>
<td>IQConnID</td>
<td>The Sybase IQ specific connection ID</td>
</tr>
<tr>
<td>Userid</td>
<td>The name of the user that opened the connection “ConnHandle”</td>
</tr>
<tr>
<td>BlockedOn</td>
<td>The connection on which a particular connection is blocked; 0 if not blocked on any connection</td>
</tr>
<tr>
<td>BlockUserid</td>
<td>The owner of the blocking connection; NULL if there is no blocking connection</td>
</tr>
<tr>
<td>ReqType</td>
<td>The type of the request made through the connection; DO NOTHING if no command is issued</td>
</tr>
<tr>
<td>IQCmdType</td>
<td>The type of Sybase IQ command issued from the connection; NONE if no command is issued</td>
</tr>
<tr>
<td>IQIdle</td>
<td>The time in seconds since the last Sybase IQ command was issued through the connection; in case of no last Sybase IQ command, the time since ‘01-01-2000’ is displayed</td>
</tr>
<tr>
<td>SAIdle</td>
<td>The time in seconds since the last SA request was issued through the connection; in case of no last SA command, the time since ‘01-01-2000’ is displayed</td>
</tr>
<tr>
<td>IQCursors</td>
<td>The number of active cursors in the connection; 0 if no cursors</td>
</tr>
<tr>
<td>IQThreads</td>
<td>The number of threads with the connection. At least one thread is started as soon as the connection is opened, so the minimum value for IQThreads is 1.</td>
</tr>
<tr>
<td>TempTableSpaceKB</td>
<td>The size of temporary table space in kilobytes; 0 if no temporary table space is used</td>
</tr>
<tr>
<td>TempWorkSpaceKB</td>
<td>The size of temporary workspace in kilobytes; 0 if no temporary workspace is used</td>
</tr>
</tbody>
</table>

**Adaptive Server Enterprise compatibility** The Sybase IQ sp_iqwho stored procedure incorporates the Sybase IQ equivalents of columns displayed by the Adaptive Server Enterprise sp_who procedure. Some Adaptive Server Enterprise columns are omitted, as they are not applicable to Sybase IQ. Table 7-70 maps the Adaptive Server Enterprise sp_who columns to the columns displayed by sp_iqwho.
Table 7-70: Mapping of sp_who and sp_iqwho columns

<table>
<thead>
<tr>
<th>sp_who column</th>
<th>sp_iqwho column</th>
</tr>
</thead>
<tbody>
<tr>
<td>fid</td>
<td>Family to which a lock belongs; omitted, as not applicable to Sybase IQ</td>
</tr>
<tr>
<td>spid</td>
<td>ConnHandle, IQConnID</td>
</tr>
<tr>
<td>status</td>
<td>IQIdle, SAIdle</td>
</tr>
<tr>
<td>loginame</td>
<td>Userid</td>
</tr>
<tr>
<td>origname</td>
<td>User alias; omitted, as not applicable to Sybase IQ</td>
</tr>
<tr>
<td>hostname</td>
<td>Name of the host on which the server is running; currently not supported</td>
</tr>
<tr>
<td>blk_spid</td>
<td>BlockedOn</td>
</tr>
<tr>
<td>dbname</td>
<td>Omitted, as there is one server and one database for Sybase IQ and they are the same for every connection</td>
</tr>
<tr>
<td>cmd</td>
<td>ReqType, IQCmdType</td>
</tr>
<tr>
<td>block_xloid</td>
<td>BlockUserid</td>
</tr>
</tbody>
</table>

### Usage

**conhandle**  An integer representing the connection ID. If this parameter is specified, sp_iqwho returns information only about the specified connection. If the specified connection is not open, no rows are displayed in the output.

**user-name**  A char(255) parameter representing a user login name. If this parameter is specified, sp_iqwho returns information only about the specified user. If the specified user has not opened any connections, no rows are displayed in the output. If the specified user name does not exist in the database, sp_iqwho returns the error message "User user-name does not exist"

**arg-type**  The arg-type parameter is optional and can be specified only when the first parameter has been specified. The only value for arg-type is "user". If the arg-type value is specified as "user", sp_iqwho interprets the first parameter as a user name, even if the first parameter is numeric. If any value other than "user" is specified for arg-type, sp_iqwho returns the error "Invalid parameter."

Enclose the arg-type value in double quotes.

If no parameters are specified, sp_iqwho displays information about all currently active connections and users.

Either a connection handle or a user name can be specified as the first sp_iqwho parameter. The parameters conhandle and user-name are exclusive and optional. Only one of these parameters can be specified at a time. By default, if the first parameter is numeric, the parameter is assumed to be a connection handle. If the first parameter is not numeric, it is assumed to be a user name.
Sybase IQ allows numeric user names. The arg-type parameter directs sp_iqwho to interpret a numeric value in the first parameter as a user name. For example:

\[ \text{sp_iqwho 1, "user"} \]

When the arg-type "user" is specified, sp_iqwho interprets the first parameter as a user name, not as a connection ID. If a user named 1 exists in the database, sp_iqwho displays information about connections opened by user 1.

**Table 7-71: sp_iqwho usage examples**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_iqwho</td>
<td>Displays all active connections</td>
</tr>
<tr>
<td>sp_iqwho 3</td>
<td>Displays information about connection 3</td>
</tr>
<tr>
<td>sp_iqwho “DBA”</td>
<td>Displays connections opened by user DBA</td>
</tr>
<tr>
<td>sp_iqwho 3, “user”</td>
<td>Interprets 3 as a user name and displays connections opened by user 3.</td>
</tr>
<tr>
<td></td>
<td>If user 3 does not exist, returns the error &quot;User 3 does not exist&quot;</td>
</tr>
<tr>
<td>sp_iqwho non-existing-user</td>
<td>Returns error &quot;User non-existing-user does not exist&quot;</td>
</tr>
<tr>
<td>sp_iqwho 3, “xyz”</td>
<td>Returns the error &quot;Invalid parameter: xyz&quot;</td>
</tr>
</tbody>
</table>

**Example**

Display all active connections:

<table>
<thead>
<tr>
<th>ConnHandle</th>
<th>IQConnID</th>
<th>Userid</th>
<th>ReqType</th>
<th>IQCmdType</th>
<th>BlockedOn</th>
<th>BlockUserid</th>
<th>IQCursors</th>
<th>IQThreads</th>
<th>IQIdle</th>
<th>SAIdle</th>
<th>TempTableSpaceKB</th>
<th>TempWorkSpaceKB</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>118</td>
<td>DBA</td>
<td>CURSOR_OPEN</td>
<td>IQUTILITYOPENCURSOR</td>
<td>0</td>
<td>(NULL)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>119</td>
<td>shweta</td>
<td>DO NOTHING</td>
<td>NONE</td>
<td>0</td>
<td>(NULL)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>16238757</td>
<td>470</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also

“sp_iqconnection procedure” on page 373

“sa_conn_info system procedure” on page 494

**sp_iqworkmon procedure**

Function

Controls collection of workload monitor usage information, and reports monitoring collection status.

**Syntax**

\[ \text{sp_iqworkmon [ 'action' ] [ , 'mode' ]} \]

action = ‘start’, ‘stop’, ‘status’, ‘reset’

mode = ‘index’, ‘table’, ‘column’, ‘all’
For example:

```
sp_iqworkmon 'start', 'all'
```

If one argument is specified, it can only be action. For example:

```
sp_iqworkmon 'stop'
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Usage**

- **action**: Specifies the control action to apply. A value of *start* starts monitoring for the specified mode immediately. A value of *stop* stops monitoring immediately. A value of *reset* clears all collected usage information. A value of *status* (the default) displays the current status without changing state.

- **mode**: Specifies the type of monitoring to control. The INDEX, TABLE, and COLUMN keywords individually control monitoring of index usage, table usage, and column usage respectively. The default ALL keyword controls monitoring of all usage monitoring features simultaneously.

There is always a result set when you execute `sp_iqworkmon`. If you specify a specific mode (such as index), only the row for that mode appears.

Usage is collected only for SQL statements containing a FROM clause; for example, SELECT, UPDATE, and DELETE.

**Table 7-72: sp_iqworkmon columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonMode</td>
<td>Table, index, or column</td>
</tr>
<tr>
<td>Status</td>
<td>Started or stopped</td>
</tr>
<tr>
<td>Rowcount</td>
<td>Current number of rows collected</td>
</tr>
</tbody>
</table>

**Example**

Sample output from the `sp_iqworkmon` procedure:

```
MonMode Status Rowcount
--- --- ----
index started 15
table started 10
column started 31
```

**See also**


Catalog stored procedures

The following catalog store stored procedures return result sets displaying database server, database, and connection properties in tabular form. These procedures are owned by the dbo user ID. The PUBLIC group has EXECUTE permission on them.

**sa_audit_string system procedure**

*Function*  
Adds a string to the transaction log.

*Syntax*  
```
sa_audit_string ( string )
```

*Permissions*  
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

*Description*  
If auditing is turned on, this system procedure adds a comment into the audit log. The string can be a maximum of 200 bytes long.

*Example*  
The following call adds a comment into the audit log:
```
CALL sa_audit_string( 'Auditing test' )
```

**sa_checkpoint_execute system procedure**

*Function*  
Allows the execution of shell commands during a checkpoint.

*Syntax*  
```
sa_checkpoint_execute 'shell_commands'
```

*Parameters*  
**shell_commands**  
One or more user commands to be executed in a system shell. The shell commands are specific to the system shell. Commands are separated by a semicolon (;).

*Permissions*  
None.

*Description*  
Allows users to execute shell commands to copy a running database from the middle of a checkpoint operation, when the server is quiescent. The copied database can be started and goes through normal recovery, similar to recovery following a system failure.
sa_checkpoint_execute initiates a checkpoint, and then executes a system shell from the middle of the checkpoint, passing the user commands to the shell. The server then waits for the shell to complete, creating an arbitrary size time window during which to copy database files. Most database activity stops while the checkpoint is executing, so the duration of the shell commands should be limited to acceptable user response time.

If the shell commands return a nonzero status, sa_checkpoint_execute returns an error.

Do not use the sa_checkpoint_execute with interactive commands, as the server must wait until the interactive command is killed. Supply override flags to disable prompting for any shell commands that might become interactive; in other words, the COPY, MOVE, and DELETE commands might prompt for confirmation.

The intended use of sa_checkpoint_execute is with disk mirroring, to split mirrored devices.

When using sa_checkpoint_execute to copy iqdemo.* files to another directory, all files are copied except the .db and .log files. Error -910 is returned.

This error not a product defect but a Windows limitation; the Windows copy command cannot copy catalog files while they are open by the database.

Example

Assuming you have created a subdirectory named backup, the following statement issues a checkpoint, copies all of the iqdemo database files to the backup subdirectory, and completes the checkpoint:

\[\text{sa_checkpoint_execute 'cp iqdemo.* backup/'}\]

**sa_conn_activity system procedure**

**Function**

Returns the most recently prepared SQL statement for each connection to databases on the server.

**Syntax**

\[\text{sa_conn_activity}\]

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Side effects**

None

**Description**

The sa_conn_activity procedure returns a result set consisting of the most recently prepared SQL statement for each connection if the server has been told to collect the information. To obtain the result set, specify the -zl option when starting the database server or execute the following:
CALL sa_server_option('Remember_last_statement','ON')

This procedure is useful when the database server is busy and you want to obtain information about what SQL statement is prepared for each connection. This feature can be used as an alternative to request-level logging.

For information on the LastStatement property from which these values are derived, see *SQL Anywhere Server – Database Administration*.

For information about the -zl command line option, see Chapter 1, “Running the Database Server” in *Utility Guide*.

For information about the remember_last_statement setting, see “sa_server_option system procedure” on page 504.

---

**sa_conn_info system procedure**

**Function**

Reports connection property information.

**Syntax**

```
  sa_conn_info ( [ connection-id ] )
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**

Returns a result set consisting of the following connection properties for the supplied connection. If no `connection-id` is supplied, information for all current connections to databases on the server is returned.

- Number
- Name
- Userid
- DBNumber
- LastReqTime
- ReqType
- CommLink
- NodeAddr
- ClientPort
- ServerPort
- BlockedOn
- LockRowID
• LockIndexID
• LockTable
• UncommitOps

In a deadlock situation, the BlockedOn value returned by this procedure allows you to check which users are blocked, and who they are blocked on.

Example

```
sa_conn_info
569851433, '', 'DBA', 0, '', '0.0', 1,
'CURSOR_OPEN', 'local', '', 6821, 0, 0, 1008
```

**sa_conn_properties system procedure**

Function

Reports connection property information.

Syntax

```
sa_conn_properties ([ connection-id ])
```

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Returns the connection ID as Number, PropNum, PropName, PropDescription, and Value for each available connection property. Omitting the `connection-id` produces results for all connections.

For a listing of available connection properties, see “Connection properties” in the SQL Anywhere documentation at [SQL Anywhere 11.0.1 > SQL Anywhere Server - Database Administration > Configuring Your Database > Connection, database, and database server properties](#).

Example

```
sa_conn_properties
```

<table>
<thead>
<tr>
<th>Number</th>
<th>PropNum</th>
<th>PropName</th>
<th>PropDescription</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2</td>
<td>BytesReceived</td>
<td>Bytes received by server</td>
<td>'4157'</td>
<td></td>
</tr>
<tr>
<td>1,3</td>
<td>BytesReceivedUncomp</td>
<td>Bytes received after decompression</td>
<td>'4241'</td>
<td></td>
</tr>
<tr>
<td>1,4</td>
<td>BytesSent</td>
<td>Bytes sent to client</td>
<td>'15192'</td>
<td></td>
</tr>
<tr>
<td>1,10</td>
<td>CacheHits</td>
<td>Cache Hits</td>
<td>'1291'</td>
<td></td>
</tr>
<tr>
<td>1,11</td>
<td>CacheReadIndInt</td>
<td>Cache index interior reads</td>
<td>'58'</td>
<td></td>
</tr>
<tr>
<td>1,12</td>
<td>CacheReadIndLeaf</td>
<td>Cache index leaf reads</td>
<td>'121'</td>
<td></td>
</tr>
<tr>
<td>1,15</td>
<td>CacheRead</td>
<td>Cache reads</td>
<td>'1318'</td>
<td></td>
</tr>
<tr>
<td>1,19</td>
<td>CacheReadTable</td>
<td>Cache table reads</td>
<td>'387'</td>
<td></td>
</tr>
<tr>
<td>1,20</td>
<td>CacheReadWorkTable</td>
<td>Cache work table reads</td>
<td>'4'</td>
<td></td>
</tr>
<tr>
<td>1,21</td>
<td>CarverHeapPages</td>
<td>Cache pages used for carvers</td>
<td>'0'</td>
<td></td>
</tr>
<tr>
<td>1,39</td>
<td>ClientStmtCacheHits</td>
<td>Number of prepares not required because of</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Catalog stored procedures

the client statement cache', '0'
1,40,'ClientStmtCacheMisses', 'Number of prepares in the client
statement cache which were prepared again', '0'
1,41,'Commit', 'Number of commit requests', '1'
1,48,'Cursor', 'Declared cursors', '4'
1,41,'Commit', 'Number of commit
requests', '1'

Note To get cache hit statistics for the entire cache, use sa_eng_properties,
and see the output lines for CacheHitsEng, CacheReadEng, and DiskReadEng. If
you run the same query on the catalog store repeatedly, the first time you
should see reads increase but no cache hits; as you repeat the query, cache hits
increase in step with cache reads.

sa_db_info system procedure

Function Reports database property information.

Syntax

    sa_db_info ( [ database-id ] )

Permissions DBA authority required. Users without DBA authority must be granted
EXECUTE permission to run the stored procedure.

Description Returns a single row containing the Number, Alias, File, ConnCount,
PageSize, and LogName for the specified database.

Example

• The following statement returns a single row describing the current
database. Table 7-73 lists sample values.

    sa_db_info

    0,'iqdemo','
/sys1/users/test/sybase/IQ-15_1/demo/iqdemo.db',
1,4096, '/sys1/users/test/sybase/IQ-15_1/demo/
 iqdemo.log'

Table 7-73: sa_db_info sample values

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>Alias</td>
<td>iqdemo</td>
</tr>
<tr>
<td>File</td>
<td>C:\Documents and Settings\All Users\SybaseIQ\demo\iqdemo.db</td>
</tr>
<tr>
<td>ConnCount</td>
<td>1</td>
</tr>
<tr>
<td>PageSize</td>
<td>4096</td>
</tr>
</tbody>
</table>
**sa_db_properties system procedure**

**Function**
Reports database property information.

**Syntax**
```
sa_db_properties ([ database-id ])
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Returns the database ID number and the Number, PropNum, PropName, PropDescription, and Value, for each property returned by the `sa_db_info` system procedure.

**Example**
```
sa_db_properties
0,10,'CacheHits','Cache Hits','4660'
0,11,'CacheReadIndInt','Cache index interior reads','334'
0,12,'CacheReadIndLeaf','Cache index leaf reads','1117'
0,15,'CacheRead','Cache reads','4887'
0,19,'CacheTable','Cache table reads','2025'
0,20,'CacheWriteTable','Cache write table reads','4'
0,22,'ChkptFlush','Checkpoint flushed pages','64'
0,23,'ChkptPage','Checkpoint log page images saved','64'
0,24,'CheckpointUrgency','Checkpoint Urgency','7'
0,25,'Chkpt','Checkpoints','2'
0,26,'CheckpointLogBitmapSize','Checkpoint log bitmap size','0'
0,27,'CheckpointLogBitmapPagesWritten','Checkpoint log writes to bitmap','0'
0,28,'CheckpointLogCommitToDisk','Checkpoint log commit to disk','8'
0,29,'CheckpointLogPageInUse','Checkpoint log pages in use','62'
0,30,'CheckpointLogPagesRelocated','Checkpoint log pages relocated','0'
0,31,'CheckpointLogSavePreimage','Checkpoint log save preimage','64
```

**See also**
“sa_db_info system procedure” on page 496
Catalog stored procedures

sa_enable_auditing_type system procedure

Function
Enables auditing and specifies which events to audit.

Syntax
sa_enable_auditing_type( [string ] )

Parameters
string is a comma-delimited string.

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
sa_enable_auditing_type works with the PUBLIC.AUDITING option to enable auditing of specific types of information.

If you set the PUBLIC.AUDITING option to ON, and do not specify which type of information to audit, the default setting (all) takes effect. In this case, all types of auditing information are recorded.

If you set the PUBLIC.AUDITING option to ON, and disable all types of auditing using sa_disable_auditing_type, no auditing information is recorded. To re-establish auditing, use sa_enable_auditing_type to specify which type of information you want to audit.

If you set the PUBLIC.AUDITING option to OFF, then no auditing information is recorded, regardless of the sa_enable_auditing_type setting.

Example
To enable only option auditing:

sa_disable_auditing_type('all')

sa_enable_auditing_type('options')

See also
AUDITING option [database] in Chapter 2, “Database Options,” in Reference: Statements and Options

sa_eng_properties system procedure

Function
Reports database server property information.

Syntax
sa_eng_properties

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
Returns the PropNum, PropName, PropDescription, and Value for each available server property.
Table 7-74: sa_eng_properties result sets

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropNum</td>
<td>integer</td>
<td>The database server property number</td>
</tr>
<tr>
<td>PropName</td>
<td>varchar(255)</td>
<td>The database server property name</td>
</tr>
<tr>
<td>PropDescription</td>
<td>varchar(255)</td>
<td>The database server property description</td>
</tr>
<tr>
<td>Value</td>
<td>long varchar</td>
<td>The database server property value</td>
</tr>
</tbody>
</table>

For a listing of available database server properties, see “System functions” in SQL Anywhere Server – SQL Reference. See also “Server-level properties” in SQL Anywhere Server – Database Administration.

Examples

The following statement returns a set of available server properties:

```
call sa_eng_properties()
```

<table>
<thead>
<tr>
<th>PropNum</th>
<th>PropName</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IdleWrite</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>IdleChkPt</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

The following statement returns a set of available server properties:

```
sa_eng_properties
```

0,'ActiveReq','Active requests','1'
1,'AvailIO','Number of available I/O control blocks','255'
2,'BytesReceived','Bytes received by server','85898'
3,'BytesReceivedUncomp','Bytes received after decompression','85898'
4,'BytesSent','Bytes sent to client','145053'
5,'BytesSentUncomp','Bytes sent before compression','145053'
6,'CacheAllocated','Cache pages that have been allocated for server data structures','546'
7,'CacheFile','Cache pages used to hold data from database files','600'
8,'CacheFileDirty','Cache pages that are dirty (needing a write)','2'
9,'CacheFree','Number of cache pages not being used','270'
13,'CachePanics','Number of times the cache manager has failed to find a page to allocate','0'
14,'CachePinned','Pinned cache pages','591'
15,'CacheRead','Cache reads','456801'
16,'CacheReplacements','Cache replacements','0'
17,'CacheScavenged','Number of pages visited while scavenging for a page to allocate','1416'
18,'CacheScavenges','Number of times the cache manager has
sa_table_page_usage system procedure
Function
Reports information about the usage of database tables.
Syntax
sa_table_page_usage
Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.
Description
The results include the same information provided by the Information utility.

sa_disable_auditing_type system procedure
Function
Disables auditing of specific events.
Syntax
sa_disable_auditing_type([string])
Parameters
string is a comma-delimited string containing one or more of:
  all enables all types of auditing.
  connect enables auditing of both successful and failed connection attempts.
  connectFailed enables auditing of failed connection attempts.
  DDL enables auditing of DDL statements.
  options enables auditing of public options.
  permission enables auditing of permission checks, user checks, and setuser statements.
  permissionDenied enables auditing of failed permission and user checks.
### triggers

**Permission**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**

You can use the `sa_disable_auditing_type` system procedure to disable auditing of one or more categories of information.

Setting this option to `all` disables all auditing. You can also disable auditing by setting the `public.auditing` option to OFF.

#### sa_flush_cache system procedure

**Function**

Empties all pages in the database server cache.

**Syntax**

```
sa_flush_cache
```

**Permission**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**

Database administrators can use this procedure to empty the contents of the database server cache. This procedure affects the catalog store. It is of use in performance measurement to ensure repeatable results.

#### sa_make_object system procedure

**Function**

Used in a SQL script, ensures that a skeletal instance of an object exists before executing an ALTER statement that provides the actual definition.

**Syntax**

```
sa_make_object ( objtype, objname [, owner [, tabname ] ] )
```

- `object-type`: 'procedure' | 'function' | 'view' | 'trigger'

**Permission**

Resource authority required to create or modify database objects.

**Description**

This procedure is particularly useful in scripts or command files that are run repeatedly to create or modify a database schema. A common problem in such scripts is that the first time they are run, a CREATE statement must be executed, but subsequent times an ALTER statement must be executed. This procedure avoids the necessity of querying the system tables to find out whether the object exists.

To use the procedure, follow it by an ALTER statement that contains the entire object definition.
You can also use the `sa_make_object` system procedure to add a skeleton Web service.

```sql
CALL sa_make_object( 'service','my_web_service' )
```

Table 7-75 lists the meaning of the `sa_make_object` parameters.

<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>objtype</code></td>
<td>The type of object being created. The parameter must be one of 'procedure', 'function', 'view', 'service', or 'trigger'.</td>
</tr>
<tr>
<td><code>objname</code></td>
<td>The name of the object to be created.</td>
</tr>
<tr>
<td><code>owner</code></td>
<td>The owner of the object to be created. The default value is CURRENT USER.</td>
</tr>
<tr>
<td><code>tabname</code></td>
<td>Required only if <code>objtype</code> is 'trigger', in which case it specifies the name of the table on which the trigger is to be created.</td>
</tr>
</tbody>
</table>

**Examples**

- The following statements ensure that a skeleton procedure definition is created, define the procedure, and grant permissions on it. A command file containing these instructions can be run repeatedly against a database without error.

  ```sql
  CALL sa_make_object( 'procedure','myproc' );
  ALTER PROCEDURE myproc( in p1 int, in p2 char(30) )
  BEGIN
    // ...
  END;
  GRANT EXECUTE ON myproc TO public;
  ```

- The following example uses the `sa_make_object` system procedure to add a skeleton Web service.

  ```sql
  CALL sa_make_object( 'service','my_web_service' )
  ```

**See also**

“`sa_db_info` system procedure” on page 496

---

### `sa_rowgenerator` system procedure

**Function**

Returns a result set with rows between a specified start and end value.

**Syntax**

```sql
sa_rowgenerator ( [ rstart [, rend [, rstep ]] ])
```

**Parameters**

- `rstart` This optional integer parameter specifies the starting value. The default value is 0.
• **rend**  This optional integer parameter specifies the ending value. The default value is 100.

• **rstep**  This optional integer parameter specifies the increment by which the sequence values are increased. The default value is 1.

## Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

## Result sets

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_num</td>
<td>integer</td>
<td>Sequence number.</td>
</tr>
</tbody>
</table>

## Remarks

The **sa_rowgenerator** procedure can be used in the **FROM** clause of a query to generate a sequence of numbers. This procedure is an alternative to using the **RowGenerator** system table. You can use **sa_rowgenerator** for such tasks as:

• Generating test data for a known number of rows in a result set.

• Generating a result set with rows for values in every range. For example, you can generate a row for every day of the month, or you can generate ranges of PostalCodes.

• Generating a query that has a specified number of rows in the result set. This may be useful for testing the performance of queries.

You can emulate the behavior of the **RowGenerator** table with the following statement:

```
SELECT row_num FROM sa_rowgenerator(1255)
```

## Side effects

None

## Example

The following query returns a result set containing one row for each day of the current month:

```
SELECT dateadd(day, row_num-1,
                ymd(datepart(year,CURRENT DATE),
                datepart(month,CURRENT DATE), 1)) AS
day_of_month FROM sa_rowgenerator(1,31,1) WHERE
datepart(month,day_of_month) =
datepart(month,CURRENT DATE) ORDER BY row_num
```

The following query shows how many employees live in zip code ranges (0-9999), (10000-19999), ..., (90000-99999). Some of these ranges have no employees, which causes the warning **Null value eliminated in aggregate function (-109)**. The **sa_rowgenerator** procedure can be used to generate these ranges, even though no employees have a PostalCode in the range.

Reference: Building Blocks, Tables, and Procedures
The following example generates 10 rows of data and inserts them into the emp table:

```sql
INSERT INTO emp1(dept_id, salary, name) SELECT row_num, CAST( rand() * 1000 AS INTEGER), 'Mary' FROM sa_rowgenerator(1, 10)
```

**sa_server_option system procedure**

**Function**
Overrides a database server command line option while the database server is running.

**Syntax**
```
sa_server_option (option_name, option_value)
```

**Permissions**
DBA authority required. Users without DBA authority must be granted `EXECUTE` permission to run the stored procedure.

**Description**
Database administrators can use this procedure to override some database server options without restarting the database server.

You can reset these options:

<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable_connections</td>
<td>ON or OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>IQMsgMaxSize</td>
<td>Integer 0 – 2047 (inclusive) in megabytes</td>
<td>0</td>
</tr>
<tr>
<td>IQMsgNumFiles</td>
<td>Integer 0 – 64 (inclusive)</td>
<td>0</td>
</tr>
<tr>
<td>Liveness_timeout</td>
<td>Integer, in seconds</td>
<td>120</td>
</tr>
<tr>
<td>Main_Cache_Cemory_MB</td>
<td>1 – 4294967295 (2^{32} - 1)</td>
<td>32MB</td>
</tr>
<tr>
<td>Temp_Cache_Memory_MB</td>
<td>1 – 4294967295 (2^{32} - 1)</td>
<td>24MB</td>
</tr>
<tr>
<td>Procedure_profiling</td>
<td>ON, OFF, RESET, CLEAR</td>
<td>OFF</td>
</tr>
<tr>
<td>Profile_filter_conn</td>
<td>connection-id</td>
<td></td>
</tr>
<tr>
<td>Profile_filter_user</td>
<td>user-id</td>
<td></td>
</tr>
<tr>
<td>Quitting_time</td>
<td>Valid date and time</td>
<td></td>
</tr>
</tbody>
</table>
### disable_connections
When set to ON, no other connections are allowed to any databases on the database server.

<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remember_last_statement</td>
<td>ON or OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>Request_level_log_file</td>
<td>Filename</td>
<td></td>
</tr>
<tr>
<td>Request_level_log_size</td>
<td>File-size, in bytes</td>
<td></td>
</tr>
<tr>
<td>Request_level_logging</td>
<td>ALL, SQL, NONE, SQL+hostvars</td>
<td>NONE</td>
</tr>
<tr>
<td>Requests_for_connection</td>
<td>connection-id, -1</td>
<td></td>
</tr>
<tr>
<td>Requests_for_database</td>
<td>database-id, -1</td>
<td></td>
</tr>
</tbody>
</table>

**IQMsgMaxSize** Limits the maximum size of the message log. IQMsgMaxSize is an integer 0-2047 (inclusive), in megabytes. The default value is 0, which specifies that there is no limit on message log size. IQMsgMaxSize corresponds to the -iqmsgsz server switch and takes precedence over the value of -iqmsgsz.

A IQMsgMaxSize value $n$ greater than 0 means that the message log can grow up to $n$ megabytes in size. For example, the following statement limits the size of the message log to 50MB:

```sql
CALL sa_server_option('IQMsgMaxSize','50')
```

A -iqmsgsz value $n$ greater than 0 means that the message log can grow up to $n$ megabytes in size. For example, the following command limits the size of the message log to 100MB:

```shell
start_iq -n iqdemo iqdemo.db ... <other options> ... -iqmsgsz 100
```

For information on the -iqmsgsz server startup switch, see “Starting the database server” in Chapter 1, “Running the Database Server” of the *Utility Guide*.

For information on message log management, see “Message logging” in Chapter 1, “Overview of Sybase IQ System Administration” of the *System Administration Guide: Volume 1*.

**IQMsgNumFiles** Specifies the number of archives of the old message log maintained by the server. The value of IQMsgNumFiles takes effect only if the IQMsgMaxSize server property or the -iqmsgsz server startup switch is non-zero. IQMsgNumFiles corresponds to the -iqmsgnum server switch and takes precedence over the value of -iqmsgnum.

IQMsgNumFiles is an integer 0-64 (inclusive). The default value is 0, which means that messages are wrapped in the main message log file.
A IQMsgNumFiles value \( n \) greater than 0 means that the server maintains \( n \) message log archives. For example, the following statement specifies that the server maintain 5 archives of the message log:

```sql
CALL sa_server_option('IQMsgNumFiles','5')
```

For information on the -iqmsgnum server startup switch, see “Starting the database server” in Chapter 1, “Running the Database Server” of the Utility Guide.

For information on message log management, see “Message logging” in Chapter 1, “Overview of Sybase IQ System Administration” of the System Administration Guide: Volume 1.

**liveness_timeout**  A liveness packet is sent periodically across a client/server TCP/IP or SPX network to confirm that a connection is intact. If the network server runs for a liveness_timeout period without detecting a liveness packet, the communication is severed.

For information on the -tl command line option, see “Limiting inactive connections” in Chapter 2, “Running Sybase IQ” in System Administration Guide: Volume 1.

**main_cache_memory_mb**  Changes the default of the main shared memory buffer cache dynamically. This option can be set on a running server but cannot change cache size on a running database. If two databases need to run with different cache sizes, set the option before starting each database. If the cache size is set using the -iqmc server startup switch, Sybase IQ uses the specified value for all databases started on that server unless sa_server_option specifies a new value. For more information, see “Buffer caches and physical memory” in Chapter 4, “Managing System Resources,” in the Performance and Tuning Guide.

**procedure_profiling**  Controls procedure profiling for stored procedures, functions, events, and triggers. Procedure profiling shows you how long it takes your stored procedures, functions, events, and triggers to execute, as well as how long each line takes to execute. You can also set procedure profiling options on the Database property sheet in Sybase Central. Collected information appears on the Profile tab in the right pane of Sybase Central.

- **ON** enables procedure profiling for the database you are currently connected to.
- **OFF** disables procedure profiling and leaves the profiling data available for viewing.
- **RESET** returns the profiling counters to zero, without changing the ON or OFF setting.
• **CLEAR** returns the profiling counters to zero and disables procedure profiling.

Once profiling is enabled, you can use the `sa_procedure_profile_summary` and `sa_procedure_profile` stored procedures to retrieve profiling information from the database. For more information about these procedures, see *SQL Anywhere Server – SQL Reference*.

For more information about viewing procedure profiling information in Sybase Central, see “Profiling database procedures” in the *Performance and Tuning Guide*.

**profile_filter_conn** Instructs the database server to capture profiling information for a specific connection ID.

**profile_filter_user** Instructs the database server to capture profiling information for a specific user ID.

**quitting_time** Instructs the database server to shut down at the specified time.

For more information on quitting_time, see the `-tq` server option in Chapter 1, “Running the Database Server” in the *Utility Guide*.

**remember_last_statement** Instructs the database server to capture the most recently prepared SQL statement for each connection to databases on the server. For stored procedure calls, only the outermost procedure call appears, not the statements within the procedure.

You can obtain the current value of the remember_last_statement setting using the `RememberLastStatement` property function as follows:

```sql
SELECT property( 'RememberLastStatement' )
```

For more information, see `-zl` server option in Chapter 1, “Running the Database Server” in the *Utility Guide*.

When remember_last_statement is turned on, the following statement returns the most recently prepared statement for the specified connection.

```sql
SELECT connection_property( 'LastStatement', conn_id )
```

The stored procedure `sa_conn_activity` returns this same information for all connections.

**request_level_log_file** The name of the file used to record logging information. A name of NULL stops logging to file. Any backslash characters in the file name must be doubled, as this is a SQL string.

**request_level_log_size** The maximum size of the file used to record logging information, in bytes.
When the request-level log file reaches the size specified by either the
sa_server_option system procedure or the -zs server option, the file is renamed
with the extension .old appended (replacing an existing file with the same
name if one exists). The request-level log file is then restarted.

request_level_logging  The logging options can be ALL, SQL, NONE,
HOSTVARS, PLAN, PROCEDURES, and TRIGGERS, separated by “+”, ON and
ALL are equivalent. OFF and NONE are equivalent. This call turns on logging
of individual SQL statements sent to the database server, for use in
troubleshooting, in conjunction with the database server -zr and -zo options.

SQL includes basic SQL statement related requests. ALL includes SQL requests,
plus other requests, which can significantly increase the size of the log. ALL
also enables recording of host variable values. If TRIGGERS is specified, all
stored procedure statements (including those in triggers) are recorded in the
request log.

The settings request_level_debugging and request_level_logging are equivalent.

When you set request_level_logging to OFF, the request-level log file is
closed.

If you select SQL, only the following types of request are recorded:

• START DATABASE
• STOP ENGINE
• STOP DATABASE
• Statement preparation
• Statement execution
• EXECUTE IMMEDIATE statements
• Option settings
• COMMIT statements
• ROLLBACK statements
• PREPARE TO COMMIT operations
• Connections
• Disconnections
• Beginnings of transactions
• DROP STATEMENT statement
• Cursor explanations
• Cursor closings
• Cursor resume
• Errors

Setting request_level_logging to SQL+HOSTVARS outputs both SQL (as though you specified request_level_logging=SQL) and host variable values to the log.

You can find the current value of the request_level_logging setting using property('RequestLogging').

For more information, see the -z, -zr, -zs, -zo, and -o command line options in Chapter 1, “Running the Database Server” in the Utility Guide.

See “-zr level” on page 30 in the Utility Guide for a list of requests that are logged by SQL request-level logging. See “Logging server requests” in Chapter 14, “Troubleshooting Hints” of System Administration Guide: Volume 1 for more information on using request logging. See also “Request logging” in SQL Anywhere Server – SQL Usage > Monitoring and Improving Database Performance > Improving database performance > Other diagnostic tools and techniques.

requests_for_connection Filter the request-level logging information so that only information for a particular connection is logged. This can help reduce the size of the request-level log file when monitoring a server with many active connections or multiple databases. You can obtain the connection ID by executing the following:

```
CALL sa_conn_info()
```

To specify a specific connection to be logged once you have obtained the connection ID, execute the following:

```
CALL sa_server_option('requests_for_connection',
connection-id )
```

Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:

```
CALL sa_server_option('requests_for_connection', -1)
```

requests_for_database Filter the request-level logging information so that only information for a particular database is logged. This can help reduce the size of the request-level log file when monitoring a server with many active connections or multiple databases. You can obtain the database ID by executing the following statement when you are connected to the desired database:
To specify that only information for a particular database is to be logged, execute the following:

```
CALL sa_server_option( 'requests_for_database', database-id )
```

Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:

```
CALL sa_server_option( 'requests_for_database', -1 )
```

**temp_cache_memory_mb**  Changes the default size of the temporary shared memory buffer cache dynamically. This option can be set on a running server but cannot change cache size on a running database. If two databases need to run with different cache sizes, set the option before starting each database. If the cache size is set using the -iqtc server startup switch, Sybase IQ uses the specified value for all databases started on that server unless `sa_server_option` specifies a new value. For more information, see “Buffer caches and physical memory” in Chapter 4, “Managing System Resources,” in the *Performance and Tuning Guide*.

**Examples**

The following statement disallows new connections to the database server.

```
call sa_server_option( 'disable_connections', 'ON' )
```

The following statement changes the size of the main shared memory buffer cache:

```
call sa_server_option( 'main_cache_memory_mb', '200' )
```

You must restart the database for the new size to take effect.

**See also**

“`sa_get_request_profile` system procedure,” “`sa_get_request_times` system procedure,” and “`sa_statement_text` system procedure” in *SQL Anywhere Server – SQL Reference*.

---

**sa_set_http_header system procedure**

**Function**

Permits a Web service to set an HTTP header in the result.

**Syntax**

```
sa_set_http_header( field-name, value )
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**

```
call dbo.sa_set_http_header( 'Content-Type', 'text/html' )
```
Setting the special header field @HttpStatus sets the status code returned with the request. For example, the following command sets the status code to 404 Not Found.

\[
\text{dbo.sa_set_http_header('@HttpStatus', '404')}
\]

The body of the error message is inserted automatically. Only valid HTTP error codes can be used. Setting the status to an invalid code causes an SQL error.

See also

“sa_set_http_option system procedure” on page 511

**sa_set_http_option system procedure**

Function

Permits a Web service to set an HTTP option in the result.

Syntax

\[
\text{sa_set_http_option( option-name, value )}
\]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

Use this procedure within statements or procedures that handle Web services to set options within an HTTP result set.

Currently only one option is supported:

- **CharsetConversion** Controls whether the result set is to be automatically converted from the character set of the database to the character set of the client. The only permitted values are ON and OFF. The default value is ON.

See also

“sa_set_http_header system procedure” on page 510

**sa_validate system procedure**

Function

Validates all tables in the catalog store.

Syntax

\[
\text{sa_validate [ tbl_name, ] [ owner_name, ] [ check_type ]}
\]

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description

This procedure validates each SQL Anywhere table or index in the catalog store.

For more information, see “Validation utility (dbvalid)” in Chapter 3, “Database Administration Utilities” in the *Utility Guide*. 

Reference: Building Blocks, Tables, and Procedures
Catalog stored procedures

Table 7-76 lists the meaning of the sa_validate parameters.

**Table 7-76: sa_validate options**

<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbl_name</td>
<td>Validate only the specified table. When NULL (the default), validate all tables.</td>
</tr>
<tr>
<td>owner_name</td>
<td>Validate only the tables owned by the specified user. When NULL (the default), validate tables for all users.</td>
</tr>
<tr>
<td>check_type</td>
<td>When NULL (the default), each table is checked without additional checks. The check_type value can be one of the following: data, express, full, index, or checksum.</td>
</tr>
</tbody>
</table>

Values for the `tbl_name`, `owner_name`, and `check_type` parameters are strings and must be enclosed in quotes.

The procedure returns a single column, named Messages. If all tables are valid, the column contains:

No errors detected

**Warning!** Validate a table or the entire catalog store while no connections are making changes to the database; otherwise, spurious errors might be reported, indicating some form of database corruption even though no corruption actually exists.

**Example**

The following statement validates all of the catalog store tables with an index check owned by the DBA:

```
CALL sa_validate (owner_name='DBA', check_type = 'index')
```

**sa_verify_password system procedure**

Function Validates the password of the current user.

Syntax `sa_verify_password ( string )`

Parameters

- **string** This char(128) parameter specifies the password of the current database user.

Permissions DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.
This procedure is used by sp_password. If the password matches, the procedure simply returns. If it does not match, the error string returned by the procedure is returned.

Side effects
None.

**sp_login_environment system procedure**

The `sp_login_environment` system procedure is a SQL Anywhere system procedure. See “sp_login_environment system procedure” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures.

**sp_remote_columns system procedure**

**Function**

Produces a list of the columns on a remote table, and a description of those columns. For each column, the procedure returns its database, owner, table, column, domain ID, width, scale, and nullability.

The server must be defined with the `CREATE SERVER` statement to use this system procedure.

**Note** You cannot capture output from this procedure in a file. If you use the redirection operator, you receive the message “Cursor is restricted to Fetch Next operations.”

**Syntax**

```sql
sp_remote_columns servername [ , tablename ] [ , owner ] [ , database ]
```

**Permissions**

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**

If you are entering a `CREATE EXISTING` statement and you are specifying a column list, it might be helpful to get a list of the columns that are available on a remote table. `sp_remote_columns` produces a list of the columns on a remote table and a description of those data types.

**Example**

Gets a list of the columns in the `sysobjects` table in the production database in an Adaptive Server server named `asetest`:

```sql
sp_remote_columns asetest, sysobjects, null, production
```
### sp_remote_exported_keys system procedure

**Function**

Provides information about tables with foreign keys on a specified primary key table.

The server must be defined with the `CREATE SERVER` statement to use this system procedure.

**Syntax**

```
sp_remote_exported_keys @server_name, @sp_name
    [, @sp_owner] [, @sp_qualifier]
```

**Permissions**

DBA authority required. Users without DBA authority must be granted `EXECUTE` permission to run the stored procedure.

**Description**

The `sp_remote_exported_keys` result set includes the database, owner, table, column, and name for both the primary and the foreign key, as well as the foreign-key sequence for the foreign-key column. The result set might vary because of the underlying ODBC and JDBC calls, but information about the table and column for a foreign key is always returned.

To use `sp_remote_exported_keys`, your database must be created or upgraded using version 12.4.3 or higher of Sybase IQ.

**Parameters**

Table 7-77 lists the `sp_remote_exported_keys` parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@server_name</code></td>
<td><code>varchar</code></td>
<td>Server on which the primary-key table is located. Required.</td>
</tr>
<tr>
<td><code>@sp_name</code></td>
<td><code>varchar(30)</code></td>
<td>Table containing the primary key. Required.</td>
</tr>
<tr>
<td><code>@sp_owner</code></td>
<td><code>varchar</code></td>
<td>Owner of primary-key table. Optional.</td>
</tr>
<tr>
<td><code>@sp_qualifier</code></td>
<td><code>varchar</code></td>
<td>Database containing the primary-key table. Optional.</td>
</tr>
</tbody>
</table>

**Example**

To get information about the remote tables with foreign keys on the `sysobjects` table, in the production database, in a server named `asetest`:

```
call sp_remote_exported_keys
```
sp_remote_imported_keys system procedure

Function
Provides information about remote tables with primary keys that correspond to a specified foreign key.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax
```
sp_remote_imported_keys @server_name [, @sp_name [, @sp_owner] [, @spQualifier ]]
```

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
Foreign keys reference a row in a separate table that contains the corresponding primary key. This procedure allows you to obtain a list of the remote tables with primary keys that correspond to a particular foreign key table. The `sp_remote_imported_keys` result set includes the database, owner, table, column, and name for both the primary and the foreign key, as well as the foreign key sequence for the foreign key column. The result set might vary because of the underlying ODBC and JDBC calls, but information about the table and column for a primary key is always returned.

To use `sp_remote_exported_keys`, your database must be created or upgraded using version 12.4.3 or higher of Sybase IQ.

Parameters
Table 7-78 lists the `sp_remote_imported_keys` parameters.
Catalog stored procedures

Table 7-78: sp_remote_imported_keys parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@server_name</td>
<td>varchar</td>
<td>Server on which the foreign-key table is located. Required.</td>
</tr>
<tr>
<td>@sp_name</td>
<td>varchar(30)</td>
<td>Table containing the foreign key. Required.</td>
</tr>
<tr>
<td>@sp_owner</td>
<td>varchar</td>
<td>Owner of foreign-key table. Optional.</td>
</tr>
<tr>
<td>@sp_qualifier</td>
<td>varchar</td>
<td>Database containing the foreign-key table. Optional.</td>
</tr>
</tbody>
</table>

Example

Gets information about the tables with primary keys that correspond to a foreign key on the sysobjects table, owned by “fred”, in the asetest server:

call sp_remote_imported_keys
    (@server_name='asetest', @sp_name='sysobjects',
     @sp_qualifier='production')

See also


CREATE SERVER statement in Reference: Statements and Options

sp_remote_primary_keys system procedure

Function

Provides primary key information about remote tables using remote data access.

Syntax

```
sp_remote_primary_keys @server_name [, @table_name ]
    [, @table_owner ] [, @table_qualifier ]
```

Accepts these parameters:

- `@server_name` Selects the server on which the remote table is located.
- `@table_name` Selects the remote table.
- `@table_owner` Selects the owner of the remote table.
- `@table_qualifier` Selects the database.

Permissions

DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Side effects

None
Description
Because of differences in the underlying ODBC/JDBC calls, the information returned differs slightly in terms of the catalog/database value, depending upon the remote data access class that is specified for the server. However, the important information (for example, column name) is as expected.

Standards and compatibility
Sybase Supported by Open Client/Open Server.

sp_remote_tables system procedure
Function
Returns a list of the tables on a server.

Syntax
```sql
sp_remote_tables servername [ , tablename ] [ , owner ]
[ , table_qualifier ] [ , with_table_type ]
```

The server must be defined with the CREATE SERVER statement to use this system procedure.

Permissions
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

Description
It might be helpful when configuring your database server to get a list of the remote tables available on a particular server. `sp_remote_tables` returns a list of the tables on a server.

The procedure accepts five parameters:

- `server_name` Selects the server the remote table is located on.
- `table_name` Selects the remote table.
- `table_owner` Selects the owner of the remote table.
- `table_qualifier` Selects the database.
- `with_table_type` Selects the type of remote table. This parameter is a bit type and accepts two values, 0 (the default) and 1. You must enter the value 1 if you want the result set to include a column that lists table types.

The `with_table_type` parameter is available only for databases created in SQL Anywhere 7.0.2 and higher. If you use this parameter with an older database, the following error message is returned:

Wrong number of parameters to function 'sp_remote_tables'
If a table, owner, or database name is given, the list of tables is limited to only those that match the parameters.

**Note** You cannot capture output from this procedure in a file. If you use the redirection operator, you receive the message “Cursor is restricted to Fetch Next operations.”

**Examples**
- Lists all the Microsoft Excel worksheets available from an ODBC data source named “exce”:
  ```sql```
  sp_remote_tables excel
  ```
- Lists all the tables in the production database in an Adaptive Server Enterprise server named asetest, owned by user fred:
  ```sql```
  sp_remote_tables asetest, null, fred, production
  ```

**Standards and compatibility**
*Sybase*  
Supported by Open Client/Open Server.

**See also**
CREATE SERVER statement in *Reference: Statements and Options*

### sp_servercaps system procedure

**Function**
Displays information about a remote server’s capabilities.

The server must be defined with the CREATE SERVER statement to use this system procedure.

**Syntax**
```sql```
sp_servercaps servername
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
Sybase IQ uses capability information to determine how much of a SQL statement can be forwarded to a remote server. The system tables that contain server capabilities are not populated until after Sybase IQ connects to the remote server. This information comes from syscapability and syscapabilityname system tables. The `servername` specified must be the same server name used in the CREATE SERVER statement.

**Example**
Displays information about the remote server testiq (output has been truncated):
### Standards and compatibility

**Sybase**  
Supported by Open Client/Open Server.

### See also

CREATE SERVER statement in *Reference: Statements and Options*

---

```sql
sp_servercaps testiq
1,'Alter table with add','T'
2,'Alter table with drop','T'
3,'Owner supported','T'
4,'Primary key requires index','F'
5,'Create table constraints','T'
6,'Truncate table','T'
7,'Create index','T' 7,'Create index','T'
8,'Create unique index','T'
9,'Syscapability system table initialized','T'
10,'Subquery','T'
11,'Subquery in group by','T'
12,'Subquery in comparison','T'
13,'Subquery in exist','T'
14,'Subquery in IN','T'
15,'Subquery correlated','T'
16,'Subquery in select list','T'
17,'Subquery in update','T'
20,'Order by','T'
21,'Order by expressions','T'
22,'Order by column not in select list','T'
23,'Order by allowed in update','T'
25,'Joins','T'
26,'Outer joins','T'
27,'Full outer joins','T'
28,'Multiple outer joins','T'
29,'Logical operators in outer join','T'
30,'Outer joins mixed with normal joins','T'
31,'ANSI join syntax','T'
32,'TSQL join syntax','F'
33,'ODBC outer join syntax','F'
34,'Unrestricted ANSI ON','T'
40,'Group by','T'
41,'Group by ALL','T'
45,'Aggregates','T'
46,'Aggregates with column name','T'
50,'And','T'
51,'Or','T'
52,'Like','T'
53,'Like - TSQL','T'
54,'Distinct','T'
55,'In','T'
```

### sp_tsql_environment system procedure

**Function**
To set connection options when users connect from jConnect or Open Client applications.

**Syntax**
```
sp_tsql_environment
```

**Permissions**
DBA authority required. Users without DBA authority must be granted EXECUTE permission to run the stored procedure.

**Description**
If the connection uses the TDS communication protocol (that is, if it is an Open Client connection), `sp_login_environment` calls `sp_tsql_environment`.

This procedure sets database options so that they are compatible with default Sybase Adaptive Server Enterprise behavior.

To change the default behavior, create new procedures and alter your LOGIN_PROCEDURE option to point to these new procedures.

For more information about setting LOGIN_PROCEDURE to the name of a new procedure, see Chapter 3, “Sybase IQ as a Data Server” in the *System Administration Guide: Volume 2*.

Here is the text of `sp_tsql_environment`:

```sql
create procedure dbo.sp_tsql_environment()
begin
  if db_property('IQStore') = 'Off' then
    -- ASA datastore
    set temporary option close_on_endtrans = 'OFF'
  end if;
  set temporary option ansinull = 'OFF';
  set temporary option tsql_variables = 'ON';
  set temporary option ansi_blanks = 'ON';
  set temporary option chained = 'OFF';
  set temporary option quoted_identifier = 'OFF';
  set temporary option allow_nulls_by_default = 'OFF';
  set temporary option on_tsql_error = 'CONTINUE';
  set temporary option isolation_level = '1';
  set temporary option date_format = 'YYYY-MM-DD';
  set temporary option timestamp_format = 'YYYY-MM-DD HH:NN:SS.SSS';
  set temporary option time_format = 'HH:NN:SS.SSS';
end;
```

set temporary option date_order = 'MDY';
set temporary option escape_character = 'OFF'
end

See also

LOGIN_PROCEDURE option in Chapter 2, “Database Options,” in Reference: Statements and Options
Adaptive Server Enterprise system and catalog procedures

Adaptive Server Enterprise provides system and catalog procedures to carry out many administrative functions and to obtain system information. Sybase IQ has implemented support for some of these procedures.

System procedures are built-in stored procedures used for getting reports from and updating system tables. Catalog stored procedures retrieve information from the system tables in tabular form.

**Note** While these procedures perform the same functions as they do in Adaptive Server Enterprise and pre-version 12 Sybase IQ, they are not identical. If you have preexisting scripts that use these procedures, you might want to examine the procedures. To see the text of a stored procedure, run:

```
sp_helptext 'owner.procedure_name'
```

For all system stored procedures delivered by Sybase, the owner is dbo. To see the text of a stored procedure of the same name owned by a different user, you must specify that user, for example:

```
sp_helptext 'myname.myprocedure'
```

Adaptive Server Enterprise system procedures

Table 7-79 describes the Adaptive Server Enterprise system procedures provided in Sybase IQ.

---

Adaptive Server Enterprise system and catalog procedures

Adaptive Server Enterprise provides system and catalog procedures to carry out many administrative functions and to obtain system information. Sybase IQ has implemented support for some of these procedures.

System procedures are built-in stored procedures used for getting reports from and updating system tables. Catalog stored procedures retrieve information from the system tables in tabular form.

**Note** While these procedures perform the same functions as they do in Adaptive Server Enterprise and pre-version 12 Sybase IQ, they are not identical. If you have preexisting scripts that use these procedures, you might want to examine the procedures. To see the text of a stored procedure, run:

```
sp_helptext 'owner.procedure_name'
```

For all system stored procedures delivered by Sybase, the owner is dbo. To see the text of a stored procedure of the same name owned by a different user, you must specify that user, for example:

```
sp_helptext 'myname.myprocedure'
```

Adaptive Server Enterprise system procedures

Table 7-79 describes the Adaptive Server Enterprise system procedures provided in Sybase IQ.
Table 7-79: ASE system procedures provided in Sybase IQ

<table>
<thead>
<tr>
<th>System procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_addgroup group-name</td>
<td>Adds a group to a database</td>
</tr>
<tr>
<td>sp_addlogin userid,</td>
<td>Adds a new user account to a database</td>
</tr>
<tr>
<td>password[, defdb [,</td>
<td></td>
</tr>
<tr>
<td>deflanguage [, fullname]]</td>
<td></td>
</tr>
<tr>
<td>sp_addmessage message-num,</td>
<td>Adds user-defined messages to SYSUSERMESSAGES for use by stored procedure</td>
</tr>
<tr>
<td>message_text [,</td>
<td>PRINT and RAISERROR calls</td>
</tr>
<tr>
<td>language]</td>
<td></td>
</tr>
<tr>
<td>sp_addtype typename,</td>
<td>Creates a user-defined data type. Sybase IQ does not support IDENTITY</td>
</tr>
<tr>
<td>datatype, [, &quot;identity&quot;</td>
<td>columns.</td>
</tr>
<tr>
<td></td>
<td>nulltype]</td>
</tr>
<tr>
<td>sp_adduser userid [,</td>
<td>Adds a new user to a database</td>
</tr>
<tr>
<td>name_in_db [, grpname]]</td>
<td></td>
</tr>
<tr>
<td>sp_changegroup new-group-</td>
<td>Changes a user's group or adds a user to a group</td>
</tr>
<tr>
<td>name, userid</td>
<td></td>
</tr>
<tr>
<td>sp_dboption [dbname,</td>
<td>Displays or changes database options</td>
</tr>
<tr>
<td>optname, {true</td>
<td>false}]</td>
</tr>
<tr>
<td>sp_dropgroup group-name</td>
<td>Drops a group from a database</td>
</tr>
<tr>
<td>sp_droplogin userid</td>
<td>Drops a user from a database</td>
</tr>
<tr>
<td>sp_dropmessage message-</td>
<td>Drops user-defined messages</td>
</tr>
<tr>
<td>number [, language]</td>
<td></td>
</tr>
<tr>
<td>sp_droptype typename</td>
<td>Drops a user-defined data type</td>
</tr>
<tr>
<td>sp_droouser userid</td>
<td>Drops a user from a database</td>
</tr>
<tr>
<td>sp_getmessage message-num,</td>
<td>Retrieves stored message strings from SYSUSERMESSAGES for PRINT and RAISER</td>
</tr>
<tr>
<td>@msg-var output [,</td>
<td>ERROR statements.</td>
</tr>
<tr>
<td>language]</td>
<td></td>
</tr>
<tr>
<td>sp_helptext 'owner.object-</td>
<td>Displays the text of a system procedure or view</td>
</tr>
<tr>
<td>name'</td>
<td></td>
</tr>
<tr>
<td>sp_password caller_passwd,</td>
<td>Adds or changes a password for a user ID</td>
</tr>
<tr>
<td>new_passwd [, userid]</td>
<td></td>
</tr>
</tbody>
</table>

**Note** Procedures like sp_droouser provide minimal compatibility with Adaptive Server Enterprise stored procedures. If you are accustomed to Adaptive Server Enterprise (or Sybase IQ 11.x) stored procedures, compare their text with Sybase IQ 12 procedures before using the procedure in dbisql. To compare, use the command:

```
sp_helptext 'owner.procedure_name'
```

For system stored procedures delivered by Sybase, the owner is always dbo. To see the text of a stored procedure of the same name owned by a different user, you must specify that user, for example:
Adaptive Server Enterprise catalog procedures

Sybase IQ implements most of the Adaptive Server Enterprise catalog procedures with the exception of the *sp_column_privileges* procedure. The implemented catalog procedures are described in Table 7-80. Sybase IQ also has similar customized stored procedures for some of these Adaptive Server catalog procedures.

Table 7-80: ASE catalog procedures implemented in Sybase IQ

<table>
<thead>
<tr>
<th>ASE catalog procedure</th>
<th>Description</th>
<th>IQ procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sp_columns table-name [, table-owner ] [, table-qualifier ] [, column-name ]</code></td>
<td>Returns the data types of the specified column</td>
<td></td>
</tr>
<tr>
<td><code>sp_pkeys table-name [, table_owner ] [, table_qualifier ]</code></td>
<td>Returns primary-key information for a single table</td>
<td><code>sp_iqpkeys</code></td>
</tr>
<tr>
<td><code>sp_special_columns table_name [, table-owner ] [, table-qualifier ] [, col-type ]</code></td>
<td>Returns the optimal set of columns that uniquely identify a row in a table</td>
<td></td>
</tr>
<tr>
<td><code>sp_sproc_columns proc-name [, proc_owner ] [, proc-qualifier ] [, column-name ]</code></td>
<td>Returns information about the input and return parameters of a stored procedure</td>
<td><code>sp_iqprocparm</code></td>
</tr>
<tr>
<td><code>sp_stored_procedures [ sp-name ] [, sp-owner ] [, sp-qualifier ]</code></td>
<td>Returns information about one or more stored procedures</td>
<td><code>sp_iqprocedure</code></td>
</tr>
<tr>
<td><code>sp_tables table-name [, table-owner ] [, table-qualifier ] [, table-type ]</code></td>
<td>Returns a list of objects that can appear in a FROM clause</td>
<td></td>
</tr>
</tbody>
</table>

The following Adaptive Server Enterprise catalog procedures are not supported:

- `sp_column_privileges`
- `sp_databases`
- `sp_datatype_info`
- `sp_server_info`
SQL Anywhere supported procedures

Sybase IQ supports the SQL Anywhere procedures listed in “Alphabetical list of system procedures” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > System procedures.

Note In the section “System extended stored procedures” in SQL Anywhere Server – SQL Reference, the documentation states that users must be granted EXECUTE permission or have DBA authority. Several of the subsequent procedures, however, show permissions listed as none.

The procedure below requires DBA authority.

- xp_cmdshell – system extended procedure that allows a database server to execute external shell commands
ABOUT THIS CHAPTER

This chapter lists predefined views for Sybase IQ system tables.

ABOUT SYSTEM VIEWS

The system tables are designed for internal use. To view the contents of the system tables, use the system views. A number of predefined system views are provided that present the information in the system tables in a readable format.

The definitions for the system views are included with their descriptions. Some of these definitions are complicated, but you do not need to understand them to use the views. They serve as good examples of what can be accomplished using the SELECT command and views.

ABOUT CONSOLIDATED VIEWS

Consolidated views provide data in a form more frequently required by users. For example, consolidated views often provide commonly needed joins. Consolidated views differ from system views in that they are not just a straightforward view of raw data in an underlying system table. For example, many of the columns in the system views are unintelligible ID values, whereas in the consolidated views, they are readable names.

Consolidated views such as SYSCATALOG and SYSINDEXES are common to both Sybase IQ and SQL Anywhere. For definitions of these and other consolidated views, see “Consolidated views” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views.

ABOUT COMPATIBILITY VIEWS

Compatibility views are deprecated views provided for compatibility with earlier versions of SQL Anywhere and Sybase IQ. Where possible, use system views and consolidated views instead of compatibility views, as support for compatibility views may be eliminated in future versions of Sybase IQ.

For detailed information on compatibility views, see “Compatibility views” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views.

ABOUT ASE T-SQL COMPATIBILITY VIEWS

Sybase IQ provides a set of views owned by the special user DBO, which correspond to the Adaptive Server Enterprise system tables and views. See “Transact-SQL compatibility views” on page 587.
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSARTICLE system view</td>
<td>532</td>
</tr>
<tr>
<td>SYSARTICLECOL system view</td>
<td>532</td>
</tr>
<tr>
<td>SYSARTICLECOLS consolidated view</td>
<td>532</td>
</tr>
<tr>
<td>SYSARTICLES consolidated view</td>
<td>532</td>
</tr>
<tr>
<td>SYSCAPABILITIES consolidated view</td>
<td>533</td>
</tr>
<tr>
<td>SYSCAPABILITY system view</td>
<td>533</td>
</tr>
<tr>
<td>SYSCAPABILITYNAME system view</td>
<td>533</td>
</tr>
<tr>
<td>SYSCATALOG consolidated view</td>
<td>534</td>
</tr>
<tr>
<td>SYSCHECK system view</td>
<td>534</td>
</tr>
<tr>
<td>SYSCOLAUTH consolidated view</td>
<td>534</td>
</tr>
<tr>
<td>SYSCOLPERM system view</td>
<td>535</td>
</tr>
<tr>
<td>SYSCOLLATION compatibility view (deprecated)</td>
<td>535</td>
</tr>
<tr>
<td>SYSCOLLATIONMAPPINGS compatibility view (deprecated)</td>
<td>536</td>
</tr>
<tr>
<td>SYSCOLSTAT system view</td>
<td>536</td>
</tr>
<tr>
<td>SYSCOLSTATS consolidated view</td>
<td>536</td>
</tr>
<tr>
<td>SYSCOLUMN compatibility view (deprecated)</td>
<td>537</td>
</tr>
<tr>
<td>SYSCOLUMNS consolidated view</td>
<td>537</td>
</tr>
<tr>
<td>SYSCOLUMNS ASE compatibility view</td>
<td>537</td>
</tr>
<tr>
<td>SYSCOMMENTS ASE compatibility view</td>
<td>538</td>
</tr>
<tr>
<td>SYSCONSTRAINT system view</td>
<td>538</td>
</tr>
<tr>
<td>SYSDBFILE system view</td>
<td>538</td>
</tr>
<tr>
<td>SYSDBSPACE system view</td>
<td>539</td>
</tr>
<tr>
<td>SYSDBSPACEPERM system view</td>
<td>539</td>
</tr>
<tr>
<td>SYSDependency system view</td>
<td>539</td>
</tr>
<tr>
<td>SYSDOMAIN system view</td>
<td>540</td>
</tr>
<tr>
<td>SYESEVENT system view</td>
<td>540</td>
</tr>
<tr>
<td>SYESVENTTYPE system view</td>
<td>540</td>
</tr>
<tr>
<td>SYSEXTERNENV system view</td>
<td>541</td>
</tr>
<tr>
<td>SYSEXTERNENVOBJECT system view</td>
<td>541</td>
</tr>
<tr>
<td>SYSEXTERNLOGIN system view</td>
<td>541</td>
</tr>
<tr>
<td>SYSFILE compatibility view (deprecated)</td>
<td>542</td>
</tr>
<tr>
<td>SYSFKCOL compatibility view (deprecated)</td>
<td>542</td>
</tr>
<tr>
<td>SYSFOREIGNKEY system view</td>
<td>542</td>
</tr>
<tr>
<td>SYSFOREIGNKEY compatibility view (deprecated)</td>
<td>543</td>
</tr>
<tr>
<td>SYSFOREIGNKEYS consolidated view</td>
<td>543</td>
</tr>
<tr>
<td>SYSGROUPS consolidated view</td>
<td>544</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SYSHISTORY system view</td>
<td>544</td>
</tr>
<tr>
<td>SYSIDX system view</td>
<td>544</td>
</tr>
<tr>
<td>SYSIDXCOL system view</td>
<td>545</td>
</tr>
<tr>
<td>SYSINDEX compatibility view (deprecated)</td>
<td>545</td>
</tr>
<tr>
<td>SYSINDEXES consolidated view</td>
<td>546</td>
</tr>
<tr>
<td>SYSINDEXES ASE compatibility view</td>
<td>546</td>
</tr>
<tr>
<td>SYSINFO compatibility view (deprecated)</td>
<td>546</td>
</tr>
<tr>
<td>SYSIQBACKUPHISTORY system view</td>
<td>546</td>
</tr>
<tr>
<td>SYSIQBACKUPHISTORYDETAIL system view</td>
<td>548</td>
</tr>
<tr>
<td>SYSIQCOLUMNSN system view (deprecated)</td>
<td>549</td>
</tr>
<tr>
<td>SYSIQDBSPACE system view</td>
<td>550</td>
</tr>
<tr>
<td>SYSIQDBFILE system view</td>
<td>551</td>
</tr>
<tr>
<td>SYSIQIDX system view</td>
<td>551</td>
</tr>
<tr>
<td>SYSIQINFO system view</td>
<td>552</td>
</tr>
<tr>
<td>SYSIQJOINIDX system view</td>
<td>554</td>
</tr>
<tr>
<td>SYSIQJOININDEX system view (deprecated)</td>
<td>555</td>
</tr>
<tr>
<td>SYSIQJOINIXCOLUMN system view</td>
<td>555</td>
</tr>
<tr>
<td>SYSIQJOINIXTABLE system view</td>
<td>556</td>
</tr>
<tr>
<td>SYSIQMPXLOGINPOLICYOPTION system view</td>
<td>557</td>
</tr>
<tr>
<td>SYSIQMPXSERVER system view</td>
<td>557</td>
</tr>
<tr>
<td>SYSIQOBJECTS ASE compatibility view</td>
<td>557</td>
</tr>
<tr>
<td>SYSIQPARTITIONCOLUMN system view</td>
<td>557</td>
</tr>
<tr>
<td>SYSIQTAB system view</td>
<td>558</td>
</tr>
<tr>
<td>SYSIQTABCOL system view</td>
<td>559</td>
</tr>
<tr>
<td>SYSQTABLE system view (deprecated)</td>
<td>561</td>
</tr>
<tr>
<td>SYSIQVINDEX ASE compatibility view</td>
<td>561</td>
</tr>
<tr>
<td>SYSiXCOL compatibility view (deprecated)</td>
<td>561</td>
</tr>
<tr>
<td>SYSJAR system view</td>
<td>561</td>
</tr>
<tr>
<td>SYSJARCATEGORY system view</td>
<td>562</td>
</tr>
<tr>
<td>SYSJAVACLASS system view</td>
<td>562</td>
</tr>
<tr>
<td>SYSLOGINMAP system view</td>
<td>562</td>
</tr>
<tr>
<td>SYSLOGINPOLICY system view</td>
<td>563</td>
</tr>
<tr>
<td>SYSLOGINPOLICYOPTION system view</td>
<td>563</td>
</tr>
<tr>
<td>SYSLOGINS ASE compatibility view</td>
<td>563</td>
</tr>
<tr>
<td>SYSMVOPTION system view</td>
<td>563</td>
</tr>
<tr>
<td>SYSMVOPTIONNAME system view</td>
<td>564</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SYSOBJECT system view</td>
<td>564</td>
</tr>
<tr>
<td>SYSOBJECTS ASE compatibility view</td>
<td>564</td>
</tr>
<tr>
<td>SYSOPTION system view</td>
<td>565</td>
</tr>
<tr>
<td>SYSOPTIONS consolidated view</td>
<td>565</td>
</tr>
<tr>
<td>SYSOPTSTAT system view</td>
<td>565</td>
</tr>
<tr>
<td>SYSPARTITION system view</td>
<td>566</td>
</tr>
<tr>
<td>SYSPARTITIONKEY system view</td>
<td>566</td>
</tr>
<tr>
<td>SYSPARTITIONSCHHEME system view</td>
<td>567</td>
</tr>
<tr>
<td>SYSPHYSIDX system view</td>
<td>568</td>
</tr>
<tr>
<td>SYSPROCAUTH consolidated view</td>
<td>568</td>
</tr>
<tr>
<td>SYSPROCEDURE system view</td>
<td>568</td>
</tr>
<tr>
<td>SYSPROCPARM system view</td>
<td>569</td>
</tr>
<tr>
<td>SYSPROCPARMS consolidated view</td>
<td>569</td>
</tr>
<tr>
<td>SYSPROCPERM system view</td>
<td>569</td>
</tr>
<tr>
<td>SYSPROCS consolidated view</td>
<td>570</td>
</tr>
<tr>
<td>SYSPROXYTAB system view</td>
<td>570</td>
</tr>
<tr>
<td>SYSPUBLICATION system view</td>
<td>570</td>
</tr>
<tr>
<td>SYSPUBLICATIONS consolidated view</td>
<td>571</td>
</tr>
<tr>
<td>SYSREMARK system view</td>
<td>571</td>
</tr>
<tr>
<td>SYSREMOOTELECTION system view</td>
<td>571</td>
</tr>
<tr>
<td>SYSREMOOTELECTION2 consolidated view</td>
<td>572</td>
</tr>
<tr>
<td>SYSREMOOTELECTIONS consolidated view</td>
<td>572</td>
</tr>
<tr>
<td>SYSREMOOTELECTIONTYPE system view</td>
<td>572</td>
</tr>
<tr>
<td>SYSREMOOTETYPE system view</td>
<td>573</td>
</tr>
<tr>
<td>SYSREMOOTETYPES consolidated view</td>
<td>573</td>
</tr>
<tr>
<td>SYSREMOOTEUSER system view</td>
<td>573</td>
</tr>
<tr>
<td>SYSREMOOTEUSERS consolidated view</td>
<td>574</td>
</tr>
<tr>
<td>SYSSCHEDULE system view</td>
<td>574</td>
</tr>
<tr>
<td>SYSSCARRIER system view</td>
<td>574</td>
</tr>
<tr>
<td>SYSSSOURCE system view</td>
<td>575</td>
</tr>
<tr>
<td>SYSSQLSERVERTYPE system view</td>
<td>575</td>
</tr>
<tr>
<td>SYSSUBPARTITIONKEY system view</td>
<td>575</td>
</tr>
<tr>
<td>SYSSUBSCRIPTION system view</td>
<td>575</td>
</tr>
<tr>
<td>SYSSUBSCRIPTIONS consolidated view</td>
<td>576</td>
</tr>
<tr>
<td>SYSSYNC system view</td>
<td>576</td>
</tr>
<tr>
<td>SYSSYNC2 consolidated view</td>
<td>576</td>
</tr>
<tr>
<td>SYSSYNPCOLUMNDEFUALTS consolidated view</td>
<td>577</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SYSSYNCS consolidated view</td>
<td>577</td>
</tr>
<tr>
<td>SYSSYNCSNAPSHOT system view</td>
<td>577</td>
</tr>
<tr>
<td>SYSSYNCSNAPSHOT consolidated view</td>
<td>578</td>
</tr>
<tr>
<td>SYSSYNCSUBSCRIPTIONS consolidated view</td>
<td>578</td>
</tr>
<tr>
<td>SYSSYNCSUSERS consolidated view</td>
<td>578</td>
</tr>
<tr>
<td>SYSTAB system view</td>
<td>579</td>
</tr>
<tr>
<td>SYSTABLE compatibility view (deprecated)</td>
<td>579</td>
</tr>
<tr>
<td>SYSTABAUTH consolidated view</td>
<td>579</td>
</tr>
<tr>
<td>SYSTABCOL system view</td>
<td>580</td>
</tr>
<tr>
<td>SYSTABLEPERM system view</td>
<td>580</td>
</tr>
<tr>
<td>SYSTEXTCONFIG system view</td>
<td>580</td>
</tr>
<tr>
<td>SYSTEXTIDX system view</td>
<td>581</td>
</tr>
<tr>
<td>SYSTEXTIDXTAB system view</td>
<td>581</td>
</tr>
<tr>
<td>SYSTRIGGER system view</td>
<td>581</td>
</tr>
<tr>
<td>SYSTRIGGERS consolidated view</td>
<td>582</td>
</tr>
<tr>
<td>SYSTYPEMAP system view</td>
<td>582</td>
</tr>
<tr>
<td>SYSTYPEPERM system view</td>
<td>582</td>
</tr>
<tr>
<td>SYSTYPEASE compatibility view</td>
<td>582</td>
</tr>
<tr>
<td>SYSCASE system view</td>
<td>583</td>
</tr>
<tr>
<td>SYSCASEAUTH compatibility view (deprecated)</td>
<td>583</td>
</tr>
<tr>
<td>SYSCASEAUTHORITY system view</td>
<td>583</td>
</tr>
<tr>
<td>SYSCASERLIST compatibility view (deprecated)</td>
<td>584</td>
</tr>
<tr>
<td>SYSCASEMESSAGE system view</td>
<td>584</td>
</tr>
<tr>
<td>SYSCASEOPTIONS consolidated view</td>
<td>584</td>
</tr>
<tr>
<td>SYSCASEPERM compatibility view (deprecated)</td>
<td>585</td>
</tr>
<tr>
<td>SYSCASEPERMS compatibility view (deprecated)</td>
<td>585</td>
</tr>
<tr>
<td>SYSCASETYPE system view</td>
<td>585</td>
</tr>
<tr>
<td>SYSCASES ASE compatibility view</td>
<td>586</td>
</tr>
<tr>
<td>SYSCASES system view</td>
<td>586</td>
</tr>
<tr>
<td>SYSCASES consolidated view</td>
<td>586</td>
</tr>
<tr>
<td>SYSCASES webservice system view</td>
<td>586</td>
</tr>
<tr>
<td>Transact-SQL compatibility views</td>
<td>587</td>
</tr>
</tbody>
</table>
SYSARTICLE system view

Each row of the SYSARTICLE system view describes an article in a publication. The underlying system table for this view is ISYSARTICLE.

The SYSARTICLE view is a SQL Anywhere system view. See “SYSARTICLE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSARTICLECOL system view

Each row of the SYSARTICLECOL system view identifies a column in an article. The underlying system table for this view is ISYSARTICLECOL.

The SYSARTICLECOL view is a SQL Anywhere system view. See “SYSARTICLECOL system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSARTICLECOLS consolidated view

Each row in the view identifies a column in an article.

The SYSARTICLECOLS view is a SQL Anywhere consolidated view. See “SYSARTICLECOLS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSARTICLES consolidated view

Each row in the SYSARTICLES view describes an article in a publication.
The SYSARTICLES view is a SQL Anywhere consolidated view. See “SYSARTICLES consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSCAPABILITIES consolidated view

Each row in the SYSCAPABILITIES view describes a capability. This view gets its data from the ISYSCAPABILITY and ISYSCAPABILITYNAME system tables.

The SYSCAPABILITIES view is a SQL Anywhere consolidated view. See “SYSCAPABILITIES consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSCAPABILITY system view

Each row of the SYSCAPABILITY system view identifies a capability of a remote server. The underlying system table for this view is ISYSCAPABILITY.

The SYSCAPABILITY view is a SQL Anywhere system view. See “SYSCAPABILITY system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSCAPABILITYNAME system view

Each row in the SYSCAPABILITYNAME system view names a capability that is defined in the SYSCAPABILITY system view.
SYSCAPABILITYNAME view

The SYSCAPABILITYNAME view is a SQL Anywhere system view. See “SYSCAPABILITYNAME system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSCATALOG consolidated view

Each row in the SYSCATALOG view describes a system table.

The SYSCATALOG view is a SQL Anywhere consolidated view. See “SYSCATALOG consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSCHECK system view

Each row in the SYSCHECK system view provides the definition for a named check constraint in a table. The underlying system table for this view is ISYSCHECK.

The SYSCHECK view is a SQL Anywhere system view. See “SYSCHECK system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSCOLAUTH consolidated view

Each row in the SYSCOLAUTH view describes the set of privileges (UPDATE, SELECT, or REFERENCES) granted on a column. The SYSCOLAUTH view provides a user-friendly presentation of data in the SYSCOLPERM system view.
The SYSCOLAUTH view is a SQL Anywhere consolidated view. See “SYSCOLAUTH consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSCOLPERM system view

The GRANT statement can give UPDATE, SELECT, or REFERENCES permission to individual columns in a table. Each column with UPDATE, SELECT, or REFERENCES permission is recorded in one row of the SYSCOLPERM system view. The underlying system table for this view is ISYSCOLPERM.

The SYSCOLPERM view is a SQL Anywhere system view. See “SYSCOLPERM system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSCOLLATION compatibility view (deprecated)

The SYSCOLLATION compatibility view contains the collation sequence information for the database. It is obtainable via built-in functions and is not kept in the catalog.

The SYSCOLLATION view is a SQL Anywhere compatibility view. See “SYSCOLLATION compatibility view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Views > Compatibility views.
SYSCOLLATIONMAPPINGS compatibility view (deprecated)

The SYSCOLLATIONMAPPINGS compatibility view contains only one row with the database collation mapping. It is obtainable via built-in functions and is not kept in the catalog.

The SYSCOLLATIONMAPPINGS view is a SQL Anywhere compatibility view. See “SYSCOLLATIONMAPPINGS compatibility view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.

SYSCOLSTAT system view

The SYSCOLSTAT system view contains the column statistics, including histograms, that are used by the optimizer. The contents of this view are best retrieved using the sa_get_histogram stored procedure or the Histogram utility. The underlying system table for this view is ISYSCOLSTAT.

The underlying system table for this view is always encrypted to protect the data from unauthorized access.

The SYSCOLSTAT view is a SQL Anywhere system view. See “SYSCOLSTAT system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSCOLSTATS consolidated view

The SYSCOLSTATS view contains the column statistics that are stored as histograms and used by the optimizer.

The SYSCOLSTATS view is a SQL Anywhere consolidated view. See “SYSCOLSTATS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.
SYSCOLUMN compatibility view (deprecated)

The SYSCOLUMN view is provided for compatibility with older versions of Sybase IQ that offered a SYSCOLUMN system table. However, the previous SYSCOLUMN table has been replaced by the ISYSTABCOL system table, and its corresponding SYSTABCOL system view, which you should use instead.

The SYSCOLUMN view is a SQL Anywhere compatibility view. See “SYSCOLUMN compatibility view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.

SYSCOLUMNS consolidated view

Each row in the SYSCOLUMNS view describes one column of each table and view in the catalog.

The SYSCOLUMNS view is a SQL Anywhere consolidated view. See “SYSCOLUMNS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSCOLUMNS ASE compatibility view

This view is owned by user DBO. syscolumns contains one row for every column in every table and view, and a row for each parameter in a procedure. See Table 8-1 on page 588.
SYSCOMMENTS ASE compatibility view

This view is owned by user DBO. syscomments contains entries for each view, rule, default, trigger, table constraint, partition, procedure, computed column, function-based index key, and other forms of compiled objects. The text column contains the original definition statements. If the text column is longer than 255 bytes, the entries span rows. Each object can occupy as many as 65,025 rows. See Table 8-1 on page 588.

SYSCONSTRAINT system view

Each row in the SYSCONSTRAINT system view describes a named constraint in the database. The underlying system table for this view is ISYSCONSTRAINT.

The SYSCONSTRAINT view is a SQL Anywhere system view. See “SYSCONSTRAINT system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSDBFILE system view

Each row in the SYSDBFILE system view describes a dbspace file. The underlying system table for this view is ISYSDBFILE.

Note This view replaces the deprecated system view SYSFILE.

The SYSDBFILE view is a SQL Anywhere system view. See “SYSDBFILE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSDBSPACE system view

Each row in the SYSDBSPACE system view describes a dbspace file. The underlying system table for this view is ISYSDBSPACE.

Note This view replaces the deprecated system view SYSFILE.

The SYSDBSPACE view is a SQL Anywhere system view. See “SYSDBSPACE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSDBSPACEPERM system view

Each row in the SYSDBSPACEPERM system view describes a permission on a dbspace file. The underlying system table for this view is ISYSDBSPACEPERM.

The SYSDBSPACEPERM view is a SQL Anywhere system view. See “SYSDBFILE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSDEPENDENCY system view

Each row in the SYSDEPENDENCY system view describes a dependency between two database objects. The underlying system table for this view is ISYSDEPENDENCY.

A dependency exists between two database objects when one object references another object in its definition. For example, if the query specification for a view references a table, the view is said to be dependent on the table. The database server tracks dependencies of views on tables, views, materialized views, and columns.
SYSDOMAIN system view

The SYSDOMAIN system view records information about built-in data types (also called domains). The contents of this view do not change during normal operation. The underlying system table for this view is ISYSDOMAIN.

The SYSDOMAIN system view is a SQL Anywhere system view. See “SYSDOMAIN system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSEVENT system view

Each row in the SYSEVENT system view describes an event created with CREATE EVENT. The underlying system table for this view is ISYSEVENT.

The SYSEVENT view is a SQL Anywhere system view. See “SYSEVENT system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSEVENTTYPE system view

The SYSEVENTTYPE system view defines the system event types that can be referenced by CREATE EVENT.

The SYSEVENTTYPE view is a SQL Anywhere system view. See “SYSEVENT system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSEXTERNENV system view

Sybase IQ includes support for six external runtime environments. These include embedded SQL and ODBC applications written in C/C++, and applications written in Java, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

Each row in the SYSEXTERNENV system view describes the information needed to identify and launch each of the external environments. The underlying system table for this view is ISYSEXTERNENV.

The SYSEXTERNENV view is a SQL Anywhere system view. See “SYSEXTERNENV system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSEXTERNENVOBJECT system view

Sybase IQ includes support for six external runtime environments. These include embedded SQL and ODBC applications written in C/C++, and applications written in Java, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

Each row in the SYSEXTERNENVOBJECT system view describes an installed external object. The underlying system table for this view is ISYSEXTERNENVOBJECT.

The SYSEXTERNENVOBJECT view is a SQL Anywhere system view. See “SYSEXTERNENVOBJECT system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSEXTERNLOGIN system view

Each row in the SYSEXTERNLOGIN system view describes an external login for remote data access. The underlying system table for this view is ISYSEXTERNLOGIN.
SYSFILE compatibility view (deprecated)

The SYSEXTERNLOGIN view is a SQL Anywhere system view. See “SYSEXTERNLOGIN system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSFILE compatibility view (deprecated)

Each row in the SYSFILE system view describes a dbspace for a database. Every database consists of one or more dbspaces; each dbspace corresponds to an operating system file.

The SYSFILE view is a SQL Anywhere compatibility view. See “SYSFILE compatibility view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSFKCOL compatibility view (deprecated)

Each row of SYSFKCOL describes the association between a foreign column in the foreign table of a relationship and the primary column in the primary table. This view is deprecated; use the SYSIDX and SYSIDXCOL system views instead.

The SYSFKCOL view is a SQL Anywhere compatibility view. See “SYSFKCOL compatibility view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSFKEY system view

Each row in the SYSFKEY system view describes a foreign key constraint in the system. The underlying system table for this view is ISYSFKEY.
The SYSFKEY view is a SQL Anywhere system view. See “SYSFKEY system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

**SYSFOREIGNKEY compatibility view (deprecated)**

The SYSFOREIGNKEY view is provided for compatibility with older versions of Sybase IQ that offered a SYSFOREIGNKEY system table. However, the previous SYSFOREIGNKEY system table has been replaced by the ISYSFKEY system table, and its corresponding SYSFKEY system view, which you should use instead.

The SYSFOREIGNKEY view is a SQL Anywhere consolidated view. See “SYSFOREIGNKEY consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

**SYSFOREIGNKEYS consolidated view**

Each row in the SYSFOREIGNKEYS view describes one foreign key for each table in the catalog.

The SYSFOREIGNKEYS view is a SQL Anywhere consolidated view. See “SYSFOREIGNKEYS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

**SYSGROUP system view**

There is one row in the SYSGROUP system view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups. The underlying system table for this view is ISYSGROUP.
SYSGROUPS consolidated view

The SYSGROUP view is a SQL Anywhere system view. See “SYSGROUP system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSGROUPS consolidated view

There is one row in the SYSGROUPS view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

The SYSGROUPS view is a SQL Anywhere consolidated view. See “SYSGROUPS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSHISTORY system view

Each row in the SYSHISTORY system view records a system operation on the database, such as a database start, a database calibration, and so on. The underlying system table for this view is ISYSHISTORY.

The SYSHISTORY view is a SQL Anywhere system view. See “SYSHISTORY system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSIDX system view

Each row in the SYSIDX system view defines a logical index in the database. The underlying system table for this view is ISYSIDX.

**Note** This view replaces the deprecated system view SYSINSDEX.

The SYSIDX view is a SQL Anywhere system view. See “SYSIDX system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSIDXCOL system view

Each row in the SYSIDXCOL system view describes one column of an index described in the SYSIDX system view. The underlying system table for this view is ISYSIDXCOL.

The SYSIDXCOL view is a SQL Anywhere system view. See “SYSIDXCOL system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSINDEX compatibility view (deprecated)

The SYSINDEX view is provided for compatibility with older versions of Sybase IQ that offered a SYSINDEX system table. However, the SYSINDEX system table has been replaced by the ISYSIDX system table, and its corresponding SYSIDX system view, which you should use instead.

The SYSINDEX view is a SQL Anywhere compatibility view. See “SYSINDEX compatibility view (deprecated)” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.
SYSINDEXES consolidated view

Each row in the SYSINDEXES view describes one index in the database. As an alternative to this view, you could also use the SYSIDX and SYSIDXCOL system views.

The SYSINDEXES view is a SQL Anywhere consolidated view. See “SYSINDEXES consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSINDEXES ASE compatibility view

This view is owned by user DBO. sysindexes contains one row for each clustered index, one row for each nonclustered index, one row for each table that has no clustered index, and one row for each table that contains text or image columns. This table also contains one row for each function-based index or index created on a computed column. See Table 8-1 on page 588.

SYSINFO compatibility view (deprecated)

The SYSINFO view indicates the database characteristics, as defined when the database was created. It always contains only one row. This view is obtainable via built-in functions and is not kept in the catalog.

The SYSINFO view is a SQL Anywhere compatibility view. See “SYSINFO compatibility view (deprecated)” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.

SYSIQBACKUPHISTORY system view

This view presents group information from ISYSIQBACKUPHISTORY in a readable format. Each row in this view describes a particular backup operation that finished successfully.
The view SYSIQBACKUP projects equivalent string values for columns type, subtype, and bkp_virtual.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Column constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bu_id</td>
<td>unsigned bigint</td>
<td>NOT NULL</td>
<td>Transaction identifier of the checkpoint of the operation. Backup ID for backup operations.</td>
</tr>
<tr>
<td>bu_time</td>
<td>timestamp</td>
<td>NOT NULL</td>
<td>Time of backup operation that is recorded in backup record.</td>
</tr>
<tr>
<td>type</td>
<td>tinyint</td>
<td>NOT NULL</td>
<td>Backup type: 0 = FULL 1 = INCREMENTAL 2 = INCREMENTAL SINCE FULL</td>
</tr>
<tr>
<td>selective_type</td>
<td>tinyint</td>
<td>NOT NULL</td>
<td>Backup subtype: 0 = ALL (backs up all dbfiles) 1 = READ/WRITE ONLY (backs up all read-write files) 2 = READ ONLY (backs up a particular read-only file)</td>
</tr>
<tr>
<td>virtual_type</td>
<td>tinyint</td>
<td>NOT NULL</td>
<td>Backup virtual type: 0 = NONE 1 = DECOUPLED 2 = ENCAPSULATED</td>
</tr>
<tr>
<td>dependson_id</td>
<td>unsigned bigint</td>
<td>NULL</td>
<td>NULL for FULL backup</td>
</tr>
<tr>
<td>cmd</td>
<td>long varchar</td>
<td>NOT NULL</td>
<td>Full text of command</td>
</tr>
<tr>
<td>creator</td>
<td>char(128)</td>
<td>NOT NULL</td>
<td>User who issued backup command</td>
</tr>
<tr>
<td>version</td>
<td>unsigned int</td>
<td>NOT NULL</td>
<td>Backup version</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

Primary key (bu_id)
**SYSIQBACKGROUNDHISTORYDETAIL system view**

This view describes all the dbfile records present in the database at backup time. Each row in this view describes a particular backup operation that finished successfully. It presents group information from ISYSIQBACKGROUNDHISTORYDETAIL in a readable format. The column constraint for each column is NOT NULL.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bu_id</td>
<td>unsigned bigint</td>
<td>Transaction identifier of the checkpoint of the operation. Backup ID for backup operation</td>
</tr>
<tr>
<td>dbspace_id</td>
<td>smallint</td>
<td>The dbspace ID of which this dbfile record is associated.</td>
</tr>
<tr>
<td>dbfile_id</td>
<td>smallint</td>
<td>The dbfile ID present in dbspace during ongoing backup operation</td>
</tr>
<tr>
<td>dbspace_rwstatus</td>
<td>char(1)</td>
<td>T indicates read-write</td>
</tr>
<tr>
<td>dbspace_createid</td>
<td>unsigned bigint</td>
<td>The transaction ID of the transaction that created the dbspace</td>
</tr>
<tr>
<td>dbspace_alterid</td>
<td>unsigned bigint</td>
<td>Transaction ID that marked the dbspace RO. If not marked, then the create ID</td>
</tr>
<tr>
<td>dbspace_online</td>
<td>char(1)</td>
<td>T indicates online</td>
</tr>
<tr>
<td>dbfile_rwstatus</td>
<td>char(1)</td>
<td>T indicates read-write</td>
</tr>
<tr>
<td>dbfile_createid</td>
<td>unsigned bigint</td>
<td>The transaction ID of the transaction that created this dbfile</td>
</tr>
<tr>
<td>dbfile_alterid</td>
<td>unsigned bigint</td>
<td>The transaction ID of the transaction that last altered the read-write status of this dbfile</td>
</tr>
<tr>
<td>is_backed_up</td>
<td>char(1)</td>
<td>Indicates that the dbfile is backed up in this backup</td>
</tr>
<tr>
<td>start_block</td>
<td>unsigned bigint</td>
<td>Start block for the dbfile</td>
</tr>
<tr>
<td>num_blocks</td>
<td>unsigned bigint</td>
<td>Total number of blocks in dbfile</td>
</tr>
<tr>
<td>num_blocks_backed_up</td>
<td>unsigned bigint</td>
<td>Total number of blocks backed up</td>
</tr>
</tbody>
</table>
CHAPTER 8  System Views

Constraints on underlying system table

Primary key (bu_id, dbfile_id)

Foreign key (txn_id) references SYS.ISYSBACKUPHISTORY

SYSIQCOLUMN system view (deprecated)

SYSIQCOLUMN has been replaced by the SYSIQTABCOL system view. See “SYSIQTABCOL system view” on page 559.

SYSIQDBFILE system view

Presents group information from ISYSIQDBFILE in a readable format.

Note  This view replaces the deprecated system view SYSIQFILE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbfile_id</td>
<td>small int</td>
<td>Unique ID for the dbfile</td>
</tr>
<tr>
<td>start_block</td>
<td>rowid</td>
<td>Number of the first block</td>
</tr>
<tr>
<td>block_count</td>
<td>rowid</td>
<td>Number of blocks for this file</td>
</tr>
<tr>
<td>reserve_size</td>
<td>rowid</td>
<td>Preallocated file system space</td>
</tr>
<tr>
<td>allocated</td>
<td>char(1)</td>
<td>Defines whether the segment is</td>
</tr>
</tbody>
</table>

Reference: Building Blocks, Tables, and Procedures 549
**SYSIQDBSPACE system view**

Presents group information from ISYSIQDBSPACE in a readable format.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbspace_id</td>
<td>small int</td>
<td>Each dbspace in a database is assigned a unique number (dbspace ID)</td>
</tr>
<tr>
<td>last_modified</td>
<td>timestamp</td>
<td>Time at which the dbspace's read-write status was last modified</td>
</tr>
<tr>
<td>segment_type</td>
<td>char(8)</td>
<td>Segment type: Main, Temp or Msg</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

Foreign key (server_id) references SYS.ISYSIQMPXSERVER

Unique (server_id, file_name)
### Constraints on underlying system table

Primary key (dbspace_id)

Foreign key (dbspace_id) references SYS.ISYSDBSPACE(dbspace_id)

#### SYSIQFILE system view (deprecated)

SYSIQFILE has been replaced by the SYSIQDBFILE system view. See “SYSIQDBFILE system view” on page 549.

#### SYSIQIDX system view

Presents group information from ISYSIQIDX in a readable format. Each row in the SYSIQIDX view describes an IQ index.

**Note** This view replaces the deprecated system view SYSIQINDEX.
### Constraints on underlying system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>unsigned int</td>
<td>The table number uniquely identifies the table to which this index applies</td>
</tr>
<tr>
<td>index_id</td>
<td>unsigned int</td>
<td>Each index for one particular table is assigned a unique index number</td>
</tr>
<tr>
<td>index_type</td>
<td>char(4)</td>
<td>Index type</td>
</tr>
<tr>
<td>index_owner</td>
<td>char(4)</td>
<td>Index owner</td>
</tr>
<tr>
<td>max_key</td>
<td>unsigned int</td>
<td>For internal use</td>
</tr>
<tr>
<td>identity_location</td>
<td>hs_vdorecid</td>
<td>For internal use</td>
</tr>
<tr>
<td>identity_size</td>
<td>unsigned int</td>
<td>For internal use</td>
</tr>
<tr>
<td>identity_location_size</td>
<td>unsigned int</td>
<td>For internal use</td>
</tr>
<tr>
<td>link_table_id</td>
<td>unsigned int</td>
<td>For internal use</td>
</tr>
<tr>
<td>link_index_id</td>
<td>unsigned int</td>
<td>For internal use</td>
</tr>
<tr>
<td>delimited_by</td>
<td>varchar(1024)</td>
<td>(WD indexes only) List of separators used to parse a column’s string into the words to be stored in that column’s WD index</td>
</tr>
<tr>
<td>limit</td>
<td>unsigned int</td>
<td>(WD indexes only) Maximum word length for WD index</td>
</tr>
</tbody>
</table>

### SYSIQINFO system view

Presents group information from ISYSIQINFO in a readable format.
The ISYSIQINFO system table indicates the database characteristics as defined when the Sybase IQ database was created using `CREATE DATABASE`. It always contains only one row. The `multiplex_name` and `last_multiplex_mode` columns were added for Sybase IQ 15.0.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_time</td>
<td>TIMESTAMP NOT NULL</td>
<td>Date and time that the database was created.</td>
</tr>
<tr>
<td>update_time</td>
<td>TIMESTAMP NOT NULL</td>
<td>Date and time of the last update.</td>
</tr>
<tr>
<td>file_format_version</td>
<td>UNSIGNED INT NOT NULL</td>
<td>File format number of files for this database.</td>
</tr>
<tr>
<td>cat_format_version</td>
<td>UNSIGNED INT NOT NULL</td>
<td>Catalog format number for this database.</td>
</tr>
<tr>
<td>sp_format_version</td>
<td>UNSIGNED INT NOT NULL</td>
<td>Stored procedure format number for this database.</td>
</tr>
<tr>
<td>block_size</td>
<td>UNSIGNED INT NOT NULL</td>
<td>Block size specified for the database.</td>
</tr>
<tr>
<td>chunk_size</td>
<td>UNSIGNED INT NOT NULL</td>
<td>Number of blocks per page as determined by the block size and page size specified for the database.</td>
</tr>
<tr>
<td>file_format_date</td>
<td>CHAR(10) NOT NULL</td>
<td>Date when file format number was last changed.</td>
</tr>
<tr>
<td>dbsig</td>
<td>BINARY(136) NOT NULL</td>
<td>Used internally by catalog.</td>
</tr>
<tr>
<td>commit_txn_id</td>
<td>xact_id</td>
<td>For internal use.</td>
</tr>
<tr>
<td>rd_commit_txn_id</td>
<td>xact_id</td>
<td>For internal use.</td>
</tr>
<tr>
<td>multiplex_name</td>
<td>CHAR(128) NULL</td>
<td>Name of the multiplex that this database is a member of.</td>
</tr>
<tr>
<td>last_multiplex_mode</td>
<td>TINYINT NULL</td>
<td>(Column unused in Sybase IQ 15.1) Mode of the server that last opened the catalog read-write. One of the following values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 – Single Node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 – Reader.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 – Coordinator.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 3 – Writer.</td>
</tr>
</tbody>
</table>
SYSIQJOINIDX system view

Presents group information from ISYSIQJOINIDX in a readable format. Each row in the SYSIQJOINIDX view describes an IQ join index.

**Note**  This view replaces the deprecated system view SYSIQJOININDEX.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>joinindex_id</td>
<td>unsigned int</td>
<td>Each join index is assigned a unique number that is the primary key.</td>
</tr>
<tr>
<td>jvt_id</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>dbspace_id</td>
<td>smallint</td>
<td>ID of the dbspace.</td>
</tr>
<tr>
<td>joinindex_name</td>
<td>char(128)</td>
<td>Defines the name of the join index.</td>
</tr>
<tr>
<td>joinindex_type</td>
<td>char(12)</td>
<td>For internal use.</td>
</tr>
<tr>
<td>creator</td>
<td>unsigned int</td>
<td>The number of the user that created the join index.</td>
</tr>
<tr>
<td>join_info_location</td>
<td>hs_vdorecid</td>
<td>For internal use.</td>
</tr>
<tr>
<td>join_info_loc_size</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>join_info_size</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>block_map</td>
<td>hs_blockmapidentity</td>
<td>For internal use.</td>
</tr>
<tr>
<td>block_map_size</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>vdo</td>
<td>hs_vdoidentity</td>
<td>For internal use.</td>
</tr>
<tr>
<td>vdo_size</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>commit_txn_id</td>
<td>xact_id</td>
<td>For internal use.</td>
</tr>
<tr>
<td>txn_id</td>
<td>xact_id</td>
<td>For internal use.</td>
</tr>
<tr>
<td>valid</td>
<td>char(1)</td>
<td>Indicates whether this join index needs to be synchronized. Y indicates that it does not require synchronization, N indicates that it does require synchronization.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

Primary key (joinindex_id)

Foreign key (jvt_id) references SYS.ISYSTAB

Foreign key (dbspace_id) references SYS.ISYSDBSPACE
Foreign key (creator) references SYS.ISYSUSER
Unique (jvt_id, commit_txn_id, txn_id)

**SYSIQJOININDEX system view (deprecated)**

SYSIQJOININDEX has been replaced by the SYSIQJOINIDX system view. See “SYSIQJOINIDX system view” on page 554.

**SYSIQJOINIXCOLUMN system view**

```
ALTER VIEW "SYS"."SYSIQJOINIXCOLUMN"
    as select * from SYS.ISYSIQJOINIXCOLUMN
```

Presents group information from ISYSIQJOINIXCOLUMN in a readable format. Each row in the SYSIQJOINIXCOLUMN view describes an IQ join index.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>joinindex_id</td>
<td>unsigned bigint</td>
<td>Corresponds to a join index value in SYSIQJOINIDX.</td>
</tr>
<tr>
<td>left_table_id</td>
<td>unsigned int</td>
<td>Corresponds to a table value in SYSTAB that forms the left side of the join operation.</td>
</tr>
<tr>
<td>left_column_id</td>
<td>unsigned int</td>
<td>Corresponds to a column value in SYSTABCOL that is part of the left side of the join.</td>
</tr>
<tr>
<td>join_type</td>
<td>char(4)</td>
<td>Only value currently supported is “=”.__</td>
</tr>
<tr>
<td>right_table_id</td>
<td>unsigned int</td>
<td>Corresponds to a table value in SYSTAB that forms the right side of the join operation.</td>
</tr>
<tr>
<td>right_column_id</td>
<td>unsigned int</td>
<td>Corresponds to a column value in SYSTABCOL that is part of the right side of the join.</td>
</tr>
</tbody>
</table>
SYSIQJOINIXTABLE system view

ALTER VIEW "SYS"."SYSIQJOINIXTABLE"
as select * from SYS.SYSIQJOINIXTABLE

Presents group information from ISYSIQJOINIXTABLE in a readable format. Each row in the SYSIQJOINIXTABLE view describes an IQ join index.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>unsigned int</td>
<td>Corresponds to a table value in SYSTAB that is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>included in a join operation.</td>
</tr>
<tr>
<td>joinindex_id</td>
<td>unsigned bigint</td>
<td>Corresponds to a join index value in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSIQJOINIDX.</td>
</tr>
<tr>
<td>active</td>
<td>unsigned int</td>
<td>Defines the number of times the table is used in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the join index.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

Primary key (joinindex_id, left_table_id, left_column_id, right_table_id, right_column_id)

Foreign key (joinindex_id) references SYS.ISYSIQJOINIDX

Foreign key (left_table_id, column_id) references SYS.ISYSTABCOL

Foreign key (right_table_id, column_id) references SYS.ISYSTABCOL

Constraints on underlying system table

order_num unsigned int For internal use.
left_order_num unsigned int For internal use.
right_order_num unsigned int For internal use.
key_type char(8) Defines the type of join on the keys. ‘NATURAL’ is a natural join, ‘KEY’ is a key join, ‘ON’ is a left outer/right outer/full join.
coalesce char(1) Not used.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_num</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>left_order_num</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>right_order_num</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>key_type</td>
<td>char(8)</td>
<td>Defines the type of join on the keys. ‘NATURAL’ is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a natural join, ‘KEY’ is a key join, ‘ON’ is a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>left outer/right outer/full join.</td>
</tr>
<tr>
<td>coalesce</td>
<td>char(1)</td>
<td>Not used.</td>
</tr>
</tbody>
</table>

**Column name** | **Column type** | **Description**                  |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>unsigned int</td>
<td>Corresponds to a table value in SYSTAB that is included in a join operation.</td>
</tr>
<tr>
<td>joinindex_id</td>
<td>unsigned bigint</td>
<td>Corresponds to a join index value in SYSIQJOINIDX.</td>
</tr>
<tr>
<td>active</td>
<td>unsigned int</td>
<td>Defines the number of times the table is used in the join index.</td>
</tr>
</tbody>
</table>
Primary key (table_id, joinindex_id)
Foreign key (table_id) references SYS.ISYSTAB
Foreign key (joinindex_id) references SYS.ISYSIQJOINIDX

**SYSIQMPXLOGINPOLICYOPTION system view**

See “SYSIQMPXLOGINPOLICYOPTION system view” in Appendix A, “Multiplex Reference,” in *Using Sybase IQ Multiplex*.

**SYSIQMPXSERVER system view**


**SYSIQOBJECTS ASE compatibility view**

SYSIQOBJECTS presents one row for each system table, user table, view, procedure, trigger, event, join index, constraint, domain (sysdomain), domain (sysusertype), column, and index. This view is owned by user DBO. See Table 8-1 on page 588.

**SYSIQPARTITIONCOLUMN system view**

```
ALTER VIEW "SYS"."SYSIQPARTITIONCOLUMN"
    as select * from SYS.ISYSIQPARTITIONCOLUMN
```
**SYSIQTAB system view**

Presents group information from ISYSIQPARTITIONCOLUMN in a readable format. Each row in the SYSIQPARTITIONCOLUMN view describes a column in a partition described in the SYSIQPARTITION view in a partitioned table described in the SYSPIARTITIONSCHEME view. SYSIQPARTITIONCOLUMN only describes partitions of columns that are not stored on the dbspace of the partition.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitioned_object_id</td>
<td>unsigned bigint</td>
<td>Unique ID assigned to each partitioned object (table)</td>
</tr>
<tr>
<td>partition_id</td>
<td>unsigned int</td>
<td>Identifies a partition in a partitioned table.</td>
</tr>
<tr>
<td>column_id</td>
<td>unsigned int</td>
<td>The column ID of the column.</td>
</tr>
<tr>
<td>dbspace_id</td>
<td>smallint</td>
<td>The dbspace ID of the dbspace where this column of the partition is stored.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

Primary key (partitioned_object_id, partition_id, column_id)

Foreign key (partitioned_object_id, partition_id) references SYS.ISYSPARTITION

Foreign key (dbspace_id) references SYS.ISYSDBSPACE

---

**SYSIQTAB system view**

```
ALTER VIEW "SYS"."SYSIQTAB"
  as select * from SYS.ISYSIQTAB
```

Presents group information from ISYSIQTAB in a readable format. Each row in the SYSIQTAB view describes an IQ table.

**Note** This view replaces the deprecated system view SYSIQTABLE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>unsigned int</td>
<td>Each table is assigned a unique number (the table number) that is the primary key.</td>
</tr>
<tr>
<td>block_map</td>
<td>hs_blockmapidentity</td>
<td>For internal use.</td>
</tr>
</tbody>
</table>
CHAPTER 8  System Views

Constraints on underlying system table

Primary key (table_id)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block_map_size</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>vdo</td>
<td>hs_vdoidentity</td>
<td>For internal use.</td>
</tr>
<tr>
<td>vdoi_size</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>info_location</td>
<td>hs_vdorecid</td>
<td>Not used. Always zero.</td>
</tr>
<tr>
<td>info_recid_size</td>
<td>unsigned int</td>
<td>Not used. Always zero.</td>
</tr>
<tr>
<td>info_location_size</td>
<td>unsigned int</td>
<td>Not used. Always zero.</td>
</tr>
<tr>
<td>commit_txn_id</td>
<td>xact_id</td>
<td>For internal use.</td>
</tr>
<tr>
<td>txn_id</td>
<td>xact_id</td>
<td>For internal use.</td>
</tr>
<tr>
<td>join_id</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>update_time</td>
<td>timestamp</td>
<td>Last date and time the IQ table was modified.</td>
</tr>
</tbody>
</table>

**SYSIQTABCOL system view**

```
ALTER VIEW "SYS"."SYSIQTABCOL"
    as select * from SYS.ISYSIQTABCOL
```

Presents group information from ISYSIQTABCOL in a readable format. Each row in the SYSIQTABCOL view describes a column in an IQ table.

**Note**  This view replaces the deprecated system view SYSIQCOLUMN.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>unsigned int</td>
<td>The table number uniquely identifies the table to which this column belongs. It corresponds to the table_id column of SYSTAB.</td>
</tr>
</tbody>
</table>
## SYSIQTABCOL system view

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_id</td>
<td>unsigned int</td>
<td>Each table starts numbering columns at 1. The order of column numbers determines the order that columns are displayed in the command <code>select * from table.</code></td>
</tr>
<tr>
<td>link_table_id</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>link_column_id</td>
<td>unsigned int</td>
<td>For internal use.</td>
</tr>
<tr>
<td>max_length</td>
<td>unsigned int</td>
<td>Indicates the maximum length allowed by the column.</td>
</tr>
<tr>
<td>approx_unique_count</td>
<td>rowid</td>
<td>Approximate number of unique values (cardinality) of this column.</td>
</tr>
<tr>
<td>cardinality</td>
<td>rowid</td>
<td>The actual number of unique values (cardinality) of this column.</td>
</tr>
<tr>
<td>has_data</td>
<td>char(1)</td>
<td>Indicates that the column contains data (T/F).</td>
</tr>
<tr>
<td>has_original</td>
<td>char(1)</td>
<td>Indicates the join index has the original data (T/F).</td>
</tr>
<tr>
<td>original_not_null</td>
<td>char(1)</td>
<td>Indicates the join index column with the original data was NOT NULL (T/F).</td>
</tr>
<tr>
<td>original_unique</td>
<td>char(1)</td>
<td>Indicates the join index column with the original data was UNIQUE (T/F).</td>
</tr>
</tbody>
</table>

### Constraints on underlying system table

Primary key (table_id)
SYSIQTABLE system view (deprecated)

SYSIQTABLE has been replaced by the SYSIQTAB system view. See “SYSIQTAB system view” on page 558.

SYSIQVINDEX ASE compatibility view

SYSIQVINDEX provides one row for each non-FP IQ index. This view is owned by user DBO. See Table 8-1 on page 588.

SYSIXCOL compatibility view (deprecated)

The SYSIXCOL view is provided for compatibility with older versions of Sybase IQ that offered a SYSIXCOL system table. However, the SYSIXCOL system table has been replaced by the ISYSIDXCOL system table, and its corresponding SYSIDXCOL system view.

The SYSIXCOL view is a SQL Anywhere compatibility view. See “SYSIXCOL compatibility view (deprecated)” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.

SYSJAR system view

Each row in the SYSJAR system view defines a JAR file stored in the database. The underlying system table for this view is ISYSJAR.

The SYSJAR view is a SQL Anywhere system view. See “SYSJAR system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSJARCOMPONENT system view

Each row in the SYSJARCOMPONENT system view defines a JAR file component. The underlying system table for this view is ISYSJARCOMPONENT.

The SYSJARCOMPONENT view is a SQL Anywhere system view. See “ISYSJARCOMPONENT system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSJAVACLASS system view

Each row in the SYSJAVACLASS system view describes one Java class stored in the database. The underlying system table for this view is ISYSJAVACLASS.

The SYSJAVACLASS view is a SQL Anywhere system view. See “SYSJAVACLASS system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSLOGINMAP system view

The SYSLOGINMAP system view contains one row for each user that can connect to the database using either an integrated login, or a Kerberos login. As a security measure, only users with DBA authority can view the contents of this view. The underlying system table for this view is ISYSLOGINMAP.

The SYSLOGINMAP view is a SQL Anywhere system view. See “SYSLOGINMAP system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSLOGINPOLICY system view

The underlying system table for this view is ISYSLOGINPOLICY.

The SYSLOGINPOLICY view is a SQL Anywhere system view. See “SYSLOGINPOLICY system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSLOGINPOLICYOPTION system view

The underlying system table for this view is ISYSLOGINPOLICYOPTION.

The SYSLOGINPOLICYOPTION view is a SQL Anywhere system view. See “SYSLOGINPOLICYOPTION system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSLOGINS ASE compatibility view

This view is owned by user DBO. syslogins contains one row for each valid Adaptive Server user account. See Table 8-2 on page 590.

SYSMVOPTION system view

Each row in the SYSMVOPTION system view describes the setting of one option value for a materialized view at the time of its creation. However, the description does not contain the option name. The underlying system table for this view is ISYSMVOPTION.

The SYSMVOPTION view is a SQL Anywhere system view. See “SYSMVOPTION system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSMVOPTIONNAME system view

Each row in the SYSMVOPTIONNAME system view contains the name of an option defined in the SYSMVOPTION system view. The underlying system table for this view is ISYSMVOPTIONNAME.

The SYSMVOPTIONNAME view is a SQL Anywhere system view. See “SYSMVOPTIONNAME system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSOBJECT system view

Each row in the SYSOBJECT system view describes a database object. The underlying system table for this view is ISYSOBJECT.

The SYSOBJECT view is a SQL Anywhere system view. See “SYSOBJECT system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSOBJECTS ASE compatibility view

This view is owned by user DBO. sysobjects contains one row for each table, view, stored procedure, extended stored procedure, log, rule, default, trigger, check constraint, referential constraint, computed column, function-based index key, and temporary object, and other forms of compiled objects. It also contains one row for each partition condition ID when object type is N. See Table 8-1 on page 588.
CHAPTER 8    System Views

SYSOPTION system view
The SYSOPTION system view contains one row for each option setting stored in the database. Each user can have their own setting for a given option. In addition, settings for the PUBLIC user ID define the default settings to be used for users that do not have their own setting. The underlying system table for this view is ISYSOPTION.

The SYSOPTION view is a SQL Anywhere system view. See “SYSOPTION system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSOPTIONS consolidated view
Each row in the SYSOPTIONS view describes one option that was created using the SET command. Each user can have their own setting for each option. In addition, settings for the PUBLIC user define the default settings to be used for users that do not have their own setting.

The SYSOPTIONS view is a SQL Anywhere consolidated view. See “SYSOPTIONS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSOPTSTAT system view
The SYSOPTSTAT system view stores the cost model calibration information as computed by the ALTER DATABASE CALIBRATE statement. The contents of this view are for internal use only and are best accessed via the sa_get_dtt system procedure. The underlying system table for this view is ISYSOPTSTAT.

The SYSOPTSTAT view is a SQL Anywhere system view. See “SYSOPTSTAT system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
**SYSPARTITION system view**

ALTER VIEW "SYS"."SYSPARTITION"
as select * from SYS.ISYSPARTITION

Presents group information from ISYSPARTITION in a readable format. Each row in the SYSPARTITION view describes a partitioned object (table or index) in the database.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitioned_object_id</td>
<td>unsigned bigint</td>
<td>Unique number assigned to each partitioned object (table)</td>
</tr>
<tr>
<td>partition_id</td>
<td>unsigned int</td>
<td>Identifies a partition in a partitioned table.</td>
</tr>
<tr>
<td>partition_object_id</td>
<td>unsigned bigint</td>
<td>Each table partition is an object itself and is assigned a unique number from the table object or index object.</td>
</tr>
<tr>
<td>partition_values</td>
<td>long varchar</td>
<td>Contains the upper bound for this range partition.</td>
</tr>
<tr>
<td>position</td>
<td>unsigned int</td>
<td>Ordinal number of partition.</td>
</tr>
<tr>
<td>partition_name</td>
<td>char(128)</td>
<td>Name of partition</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

Primary key (partitioned_object_id, partition_id)

Unique (partition_object_id, position)

Foreign key (partition_object_id) references SYS.ISYSOBJECT

Foreign key (partitioned_object_id) references SYS.ISYSOBJECT

**SYSPARTITIONKEY system view**

ALTER VIEW "SYS"."SYSPARTITIONKEY"
as select * from SYS.ISYSPARTITIONKEY

Presents group information from ISYSPARTITIONKEY in a readable format. Each row in the SYSPARTITIONKEY view describes a partitioned object (table or index) in the database.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitioned_object_id</td>
<td>unsigned bigint</td>
<td>Each partitioned object (table) is assigned a unique object number.</td>
</tr>
</tbody>
</table>
Constraints on underlying system table
Primary key (partitioned_object_id, column_id)
Foreign key (partitioned_object_id) references SYS.ISYSOBJECT

SYSPARTITIONSCHEME system view

ALTER VIEW "SYS"."SYSPARTITIONSCHEME"
as select * from SYS.ISYSPARTITIONSCHEME

Presents group information from ISYSPARTITIONSCHEME in a readable format. Each row in the SYSPARTITIONSCHEME view describes a partitioned object (table or index) in the database.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitioned_object_id</td>
<td>unsigned bigint</td>
<td>Each partitioned object (table) is assigned a unique number</td>
</tr>
<tr>
<td>partition_method</td>
<td>tinyint</td>
<td>The partitioning method for this table. Valid values: 1– for range Sybase IQ supports only range partitioning.</td>
</tr>
<tr>
<td>subpartition_method</td>
<td>tinyint</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table
Primary key (partitioned_object_id)
Foreign key (partitioned_object_id) references SYS.ISYSOBJECT
SYSPHYSIDX system view

Each row in the SYSPHYSIDX system view defines a physical index in the database. The underlying system table for this view is ISYSPHYSIDX.

The SYSPHYSIDX view is a SQL Anywhere system view. See “SYSPHYSIDX system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSPROCAUTH consolidated view

Each row in the SYSPROCAUTH view describes a set of privileges granted on a procedure. As an alternative, you can also use the SYSPROCPERM system view.

The SYSPROCAUTH view is a SQL Anywhere consolidated view. See “SYSPROCAUTH consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSPROCEDURE system view

Each row in the SYSPROCEDURE system view describes one procedure in the database. The underlying system table for this view is ISYSPROCEDURE.

The SYSPROCEDURE view is a SQL Anywhere system view. See “SYSPROCEDURE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSPROCPARM system view

Each row in the SYSPROCPARM system view describes one parameter to a procedure in the database. The underlying system table for this view is ISYSPROCPARM.

The SYSPROCPARM view is a SQL Anywhere system view. See “SYSPROCPARM system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSPROCPARMS consolidated view

Each row in the SYSPROCPARMS view describes a parameter to a procedure in the database.

The SYSPROCPARMS view is a SQL Anywhere consolidated view. See “SYSPROCPARMS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSPROCPERM system view

Each row of the SYSPROCPERM system view describes a user who is granted permission to execute a procedure. Only users who have been granted permission can execute a procedure. The underlying system table for this view is ISYSPROCPERM.

The SYSPROCPERM view is a SQL Anywhere system view. See “SYSPROCPERM system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSPROCS consolidated view

The SYSPROCS view shows the procedure or function name, the name of its creator and any comments recorded for the procedure or function.

The SYSPROCS view is a SQL Anywhere consolidated view. See “SYSPROCS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSPROXYTAB system view

Each row of the SYSPROXYTAB system view describes the remote parameters of one proxy table. The underlying system table for this view is ISYSPROXYTAB.

The SYSPROXYTAB view is a SQL Anywhere system view. See “SYSPROXYTAB system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSPUBLICATION system view

Each row in the SYSPUBLICATION system view describes a SQL Remote or MobiLink publication. The underlying system table for this view is ISYSPUBLICATION.

The SYSPUBLICATION view is a SQL Anywhere system view. See “SYSPUBLICATION system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSPUBLICATIONS consolidated view

Each row in the SYSPUBLICATIONS view describes a SQL Remote or MobiLink publication.

The SYSPUBLICATIONS view is a SQL Anywhere consolidated view. See “SYSPUBLICATIONS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSREMARK system view

Each row in the SYSREMARK system view describes a remark (or comment) for an object. The underlying system table for this view is ISYSREMARK.

The SYSREMARK view is a SQL Anywhere system view. See “SYSREMARK system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSREMOTEOPTION system view

Each row in the SYSREMOTEOPTION system view describes the value of a SQL Remote message link parameter. The underlying system table for this view is ISYSREMOTEOPTION.

Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted to users with DBA authority. The SYSREMOTEOPTION2 view provides public access to the data in this view except for the potentially sensitive columns.

The SYSREMOTEOPTION view is a SQL Anywhere system view. See “SYSREMOTEOPTION system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSREMOTEOPTION2 consolidated view

SYSREMOTEOPTION2 consolidated view

Presents, in a readable format, the columns from SYSREMOTEOPTION and SYSREMOTEOPTIONTYPE that do not contain sensitive data.

The SYSREMOTEOPTION2 view is a SQL Anywhere consolidated view. See “SYSREMOTEOPTION2 consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSREMOTEOPTIONS consolidated view

SYSREMOTEOPTIONS consolidated view

Each row of the SYSREMOTEOPTIONS view describes the values of a SQL Remote message link parameter. Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted to users with DBA authority. The SYSREMOTEOPTION2 view provides public access to the insensitive data.

The SYSREMOTEOPTIONS view is a SQL Anywhere consolidated view. See “SYSREMOTEOPTIONS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSREMOTEOPTIONTYPE system view

SYSREMOTEOPTIONTYPE system view

Each row in the SYSREMOTEOPTIONTYPE system view describes one of the SQL Remote message link parameters. The underlying system table for this view is ISYSREMOTEOPTIONTYPE.

The SYSREMOTEOPTIONTYPE view is a SQL Anywhere system view. See “SYSREMOTEOPTIONTYPE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSREMOTETYPE system view

The SYSREMOTETYPE system view contains information about SQL Remote. The underlying system table for this view is ISYSREMOTETYPE.

The SYSREMOTETYPE view is a SQL Anywhere system view. See “SYSREMOTETYPE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSREMOTETYPES consolidated view

Each row of the SYSREMOTETYPES view describes one of the SQL Remote message types, including the publisher address.

The SYSREMOTETYPES view is a SQL Anywhere consolidated view. See “SYSREMOTETYPES consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSREMOTEUSER system view

Each row in the SYSREMOTEUSER system view describes a user ID with REMOTE permissions (a subscriber), together with the status of SQL Remote messages that were sent to and from that user. The underlying system table for this view is ISYSREMOTEUSER.

The SYSREMOTEUSER view is a SQL Anywhere system view. See “SYSREMOTEUSER system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSREMOTEUSERS consolidated view

SYSREMOTEUSERS consolidated view
Each row of the SYSREMOTEUSERS view describes a user ID with REMOTE permissions (a subscriber), together with the status of SQL Remote messages that were sent to and from that user.

The SYSREMOTEUSERS view is a SQL Anywhere consolidated view. See “SYSREMOTEUSERS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSSCHEDULE system view

SYSSCHEDULE system view
Each row in the SYSSCHEDULE system view describes a time at which an event is to fire, as specified by the SCHEDULE clause of CREATE EVENT. The underlying system table for this view is ISYSSCHEDULE.

The SYSSCHEDULE view is a SQL Anywhere system view. See “SYSREMOTEUSER system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSSERVER system view

SYSSERVER system view
Each row in the SYSSERVER system view describes a remote server. The underlying system table for this view is ISYSSERVER.

The SYSSERVER view is a SQL Anywhere system view. See “SYSREMOTEUSER system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSSOURCE system view

Each row in the SYSSOURCE system view contains the source code, if applicable, for an object listed in the SYSOBJECT system view.

The SYSSOURCE view is a SQL Anywhere system view. See “SYSSOURCE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSSQLSERVERTYPE system view

The SYSSQLSERVERTYPE system view contains information relating to compatibility with Adaptive Server Enterprise. The underlying system table for this view is ISYSSQLSERVERTYPE.

The SYSSQLSERVERTYPE view is a SQL Anywhere system view. See “SYSSQLSERVERTYPE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSSUBPARTITIONKEY system view

This view is reserved for future use. Sybase IQ 15.1 does not support subpartitioning.

SYSSUBSCRIPTION system view

Each row in the SYSSUBSCRIPTION system view describes a subscription from one user ID (which must have REMOTE permissions) to one publication. The underlying system table for this view is ISYSSUBSCRIPTION.
The SYSSUBSCRIPTION view is a SQL Anywhere system view. See “SYSSUBSCRIPTION system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

### SYSSUBSCRIPTIONS consolidated view

Each row describes a subscription from one user ID (which must have REMOTE permissions) to one publication.

The SYSSUBSCRIPTIONS view is a SQL Anywhere consolidated view. See “SYSSUBSCRIPTIONS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

### SYSSYNC system view

The SYSSYNC system view contains information relating to MobiLink synchronization. Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted to users with DBA authority. The SYSSYNC2 view provides public access to the data in this view except for the potentially sensitive columns. The underlying system table for this view is ISYSSYNC.

The SYSSYNC view is a SQL Anywhere system view. See “SYSSYNC system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

### SYSSYNC2 consolidated view

The SYSSYNC2 view provides public access to the data found in the SYSSYNC system view—information relating to MobiLink synchronization—without exposing potentially sensitive data.
The SYSSYNC2 view is a SQL Anywhere consolidated view. See “SYSSYNC2 consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

**SYSSYNCPUBLICATIONDEFAULTS consolidated view**

The SYSSYNCPUBLICATIONDEFAULTS view provides the default synchronization settings associated with publications involved in MobiLink synchronization.

The SYSSYNCPUBLICATIONDEFAULTS view is a SQL Anywhere consolidated view. See “SYSSYNCPUBLICATIONDEFAULTS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

**SYSSYNCS consolidated view**

The SYSSYNCS view contains information relating to MobiLink synchronization. Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted to users with DBA authority.

The SYSSYNCS view is a SQL Anywhere consolidated view. See “SYSSYNCS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

**SYSSYNCSSCRIPT system view**

Each row in the SYSSYNCSSCRIPT system view identifies a stored procedure for MobiLink scripted upload. This view is almost identical to the SYSSYNCSSCRIPTS view, except that the values in this view are in their raw format. To see them in their human-readable format, see “SYSSYNCSSCRIPTS consolidated view”.

Reference: Building Blocks, Tables, and Procedures 577
SYSSYNSYSCONSCRIPTS consolidated view

The SYSSYNSYSCONSCRIPTS view is a SQL Anywhere system view. See “SYSSYNSYSCONSCRIPT system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSSYNSYSCONSCRIPTS consolidated view

Each row in the SYSSYNSYSCONSCRIPTS view identifies a stored procedure for MobiLink scripted upload. This view is almost identical to the SYSSYNSYSCONSCRIPT system view, except that the values are in human-readable format, as opposed to raw data.

The SYSSYNSYSCONSCRIPTS view is a SQL Anywhere consolidated view. See “SYSSYNSYSCONSCRIPTS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSSYNSYNSUBSCRIPTIONS consolidated view

The SYSSYNSYNSUBSCRIPTIONS view contains the synchronization settings associated with MobiLink synchronization subscriptions.

The SYSSYNSYNSUBSCRIPTIONS view is a SQL Anywhere consolidated view. See “SYSSYNSYNSUBSCRIPTIONS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSSYNSYNCUSERS consolidated view

A view of synchronization settings associated with MobiLink synchronization users.
The SYSSYNCUSERS view is a SQL Anywhere consolidated view. See “SYSSYNCUSERS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSTAB system view

Each row of the SYSTAB system view describes one table or view in the database. Additional information for views can be found in the SYSVIEW system view. The underlying system table for this view is ISYSTAB.

The SYSTAB view is a SQL Anywhere system view. See “SYSTAB system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSTABLE compatibility view (deprecated)

The SYSTABLE view is provided for compatibility with older versions of Sybase IQ that offered a SYSTABLE system table. However, the SYSTABLE system table has been replaced by the ISYSTAB system table, and its corresponding SYSTAB system view.

The SYSTABLE view is a SQL Anywhere compatibility view. See “SYSTABLE compatibility view (deprecated)” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.

SYSTABAUTH consolidated view

The SYSTABAUTH view contains information from the SYSTABLEPERM system view, but in a readable format.
SYSTABAUTH view

The SYSTABAUTH view is a SQL Anywhere consolidated view. See “SYSTABAUTH consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSTABCOL system view

The SYSTABCOL system view contains one row for each column of each table and view in the database. The underlying system table for this view is ISYSTABCOL.

The SYSTABCOL view is a SQL Anywhere system view. See “SYSTABCOL system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSTABLEPERM system view

Permissions given by the GRANT statement are stored in the SYSTABLEPERM system view. Each row in this view corresponds to one table, one user ID granting the permission (grantor) and one user ID granted the permission (grantee). The underlying system table for this view is ISYSTABLEPERM.

The SYSTABLEPERM view is a SQL Anywhere system view. See “SYSTABLEPERM system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSTEXTCONFIG system view

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.
The SYSTEXTCONFIG view is a SQL Anywhere system view. See “SYSTEXTCONFIG system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSTEXTIDX system view

Each row in the SYSTEXTIDX system view describes one text index. The underlying system table for this view is ISYSTEXTIDX.

The SYSTEXTIDX view is a SQL Anywhere system view. See “SYSTEXTIDX system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSTEXTIDXTAB system view

Each row in the SYSTEXTIDXTAB system view describes a generated table that is part of a text index. The underlying system table for this view is ISYSTEXTIDXTAB.

The SYSTEXTIDXTAB view is a SQL Anywhere system view. See “SYSTEXTIDXTAB system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSTRIGGER system view

Each row in the SYSTRIGGER system view describes one trigger in the database. This view also contains triggers that are automatically created for foreign-key definitions that have a referential triggered action (such as ON DELETE CASCADE). The underlying system table for this view is ISYSTRIGGER.
SYSTRIGGERS consolidated view

The SYSTRIGGER view is a SQL Anywhere system view. See “SYSTRIGGER system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSTRIGGERS consolidated view

Each row in the SYSTRIGGERS view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE).

The SYSTRIGGERS view is a SQL Anywhere consolidated view. See “SYSTRIGGERS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

SYSTYPEMAP system view

The SYSTYPEMAP system view contains the compatibility mapping values for entries in the SYSSQLSERVERTYPE system view. The underlying system table for this view is ISYSTYPEMAP.

The SYSTYPEMAP view is a SQL Anywhere system view. See “SYSTYPEMAP system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSTYPES ASE compatibility view

This view is owned by user DBO. systypes contains one row for each system-supplied and user-defined datatype. Domains (defined by rules) and defaults are given, if they exist. You cannot alter the rows that describe system-supplied datatypes. See Table 8-1 on page 588.
SYSUSER system view

Each row in the SYSUSER system view describes a user in the system. The underlying system table for this view is ISYSUSER.

The SYSUSER view is a SQL Anywhere system view. See “SYSUSER system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSUSERAUTH compatibility view (deprecated)

The SYSUSERAUTH view is provided for compatibility with older versions of Sybase IQ. Use the SYSUSERAUTHORITY system view instead.

The SYSUSERAUTH view is a SQL Anywhere compatibility view. See “SYSUSERAUTH compatibility view (deprecated)” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.

SYSUSERAUTHORITY system view

Each row of SYSUSERAUTHORITY system view describes an authority granted to one user ID. The underlying system table for this view is ISYSUSERAUTHORITY.

The SYSUSERAUTHORITY view is a SQL Anywhere system view. See “SYSUSERAUTHORITY system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
SYSUSERLIST compatibility view (deprecated)

The SYSUSERLIST view is provided for compatibility with older versions of Sybase IQ. Each row of the SYSUSERLIST view describes a user, without exposing their user_id and password. Each user is identified by their user name.

The SYSUSERLIST view is a SQL Anywhere compatibility view. See “SYSUSERLIST compatibility view (deprecated)” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.

SYSUSERMESSAGE system view

Each row in the SYSUSERMESSAGE system view holds a user-defined message for an error condition. The underlying system table for this view is ISYSUSERMESSAGE.

The SYSUSERMESSAGE view is a SQL Anywhere system view. See “SYSUSERMESSAGE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

SYSUSEROPTIONS consolidated view

The SYSUSEROPTIONS view contains the option settings that are in effect for each user. If a user has no setting for an option, this view displays the public setting for the option.

The SYSUSEROPTIONS view is a SQL Anywhere consolidated view. See “SYSUSEROPTIONS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.
SYSUSERPERM compatibility view (deprecated)

This view is deprecated because it only shows the authorities and permissions available in previous versions. Change your application to use the SYSUSERAUTHORITY system view instead. Each row of the SYSUSERPERM view describes one user ID.

The SYSUSERPERM view is a SQL Anywhere compatibility view. See “SYSUSERPERM compatibility view (deprecated)” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.

SYSUSERPERMS compatibility view (deprecated)

This view is deprecated because it only shows the authorities and permissions available in previous versions. Change your application to use the SYSUSERAUTHORITY system view instead. Similar to the SYSUSERPERM view, each row of the SYSUSERPERMS view describes one user ID. However, password information is not included. All users are allowed to read from this view.

The SYSUSERPERMS view is a SQL Anywhere compatibility view. See “SYSUSERPERMS compatibility view (deprecated)” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Compatibility views.

SYSUSERTYPE system view

Each row in the SYSUSERTYPE system view holds a description of a user-defined data type. The underlying system table for this view is ISYSUSERTYPE.

The SYSUSERTYPE view is a SQL Anywhere system view. See “SYSUSERTYPE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.
**SYSUSERS ASE compatibility view**

This view is owned by user DBO. sysusers contains one row for each user allowed in the database, and one row for each group or roles. See Table 8-1 on page 588

**SYSVIEW system view**

Each row in the SYSVIEW system view describes a view in the database. You can find additional information about views in the SYSTAB system view. The underlying system table for this view is ISYSVIEW.

You can also use the sa_materialized_view_info system procedure for a readable format of the information for materialized views.

The SYSVIEW view is a SQL Anywhere system view. See “SYSVIEW system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

**SYSVIEWS consolidated view**

Each row of the SYSVIEWS view describes one view, including its view definition.

The SYSVIEWS view is a SQL Anywhere consolidated view. See “SYSVIEWS consolidated view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > Consolidated views.

**SYSWEBSERVICE system view**

Each row in the SYSWEBSERVICE system view holds a description of a web service.
The SYSWEBSERVICE view is a SQL Anywhere system view. See “SYSWEBSERVICE system view” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Views > System views.

Transact-SQL compatibility views

Adaptive Server Enterprise and Sybase IQ have different system catalogs, reflecting the different uses for the two products.

In Adaptive Server Enterprise, there is a single master database containing a set of system tables holding information that applies to all databases on the server. Many databases may exist within the master database, and each has additional system tables associated with it.

In Sybase IQ, each database exists independently, and contains its own system tables. There is no master database that contains system information on a collection of databases. Each server may run several databases at a time, dynamically loading and unloading each database as needed.

The Adaptive Server Enterprise and Sybase IQ system catalogs are different. The Adaptive Server Enterprise system tables and views are owned by the special user dbo, and exist partly in the master database, partly in the sybsecurity database, and partly in each individual database; the Sybase IQ system tables and views are owned by the special user SYS and exist separately in each database.

To assist in preparing compatible applications, Sybase IQ provides a set of views owned by the special user dbo, which correspond to the Adaptive Server Enterprise system tables and views. Where architectural differences make the contents of a particular Adaptive Server Enterprise table or view meaningless in a Sybase IQ context, the view is empty, containing only the column names and data types.

Table 8-1, Table 8-2, and Table 8-3 list the Adaptive Server Enterprise system tables and their implementation in the Sybase IQ system catalog. The owner of all tables is dbo in each DBMS.
## Transact-SQL compatibility views

### Tables in each Adaptive Server Enterprise database

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
<th>Data?</th>
<th>Supported by IQ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysalternates</td>
<td>One row for each user mapped to a database user</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>syscolumns</td>
<td>One row for each column in a table or view, and for each parameter in a procedure. In Sybase IQ, use the owner name <code>dbo</code> when querying, i.e. <code>dbo.syscolumns</code>.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>syscomments</td>
<td>One or more rows for each view, rule, default, and procedure, giving SQL definition statement.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sysconstraints</td>
<td>One row for each referential and check constraint associated with a table or column.</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysdepends</td>
<td>One row for each procedure, view, or table that is referenced by a procedure, view.</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysindexes</td>
<td>One row for each clustered or nonclustered index, and one row for each table with no indexes, and an additional row for each table containing text or image data. In Sybase IQ, use the owner name <code>dbo</code> when querying, i.e. <code>dbo.sysindexes</code>.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sysiqobjects</td>
<td>One row for each system table, user table, view, procedure, trigger, event, join index, constraint, domain (sysdomain), domain (sysusertype), column, and index.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sysiqvindex</td>
<td>One row for each non-fp iq index.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>syskeys</td>
<td>One row for each primary, foreign, or common key; set by user (not maintained by Adaptive Server Enterprise).</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>syslogs</td>
<td>Transaction log.</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysobjects</td>
<td>One row for each table, view, procedure, rule, default, log, and (in tempdb only) temporary object. Contains compatible data only.</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Table name | Description | Data? | Supported by IQ?
--- | --- | --- | ---
sysprocedures | One row for each view, rule, default, and procedure, giving internal definition. | No | No |
sysprotects | User permissions information. | No | No |
sysreferences | One row for each referential integrity constraint declared on a table or column. | No | No |
sysroles | Maps server-wide roles to local database groups. | No | No |
syssegments | One row for each segment (named collection of disk pieces). | No | No |
systhresholds | One row for each threshold defined for the database. | No | No |
systypes | One row for each system-supplied and user-defined data type. | Yes | Yes |
sysusers | One row for each user allowed in the database. | Yes | Yes |
### Transact-SQL compatibility views

#### Tables in the Adaptive Server Enterprise master database

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
<th>Data?</th>
<th>Supported by IQ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscharsets</td>
<td>One row for each character set or sort order</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysconfigures</td>
<td>One row for each configuration parameter that can be set by a user</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>syscurconfigs</td>
<td>Information about configuration parameters currently being used by the server</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysdatabases</td>
<td>One row for each database on the server</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysdevices</td>
<td>One row for each tape dump device, disk dump device, disk for databases, and disk partition for databases</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysengines</td>
<td>One row for each server currently online</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>syslanguages</td>
<td>One row for each language (except U.S. English) known to the server</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>syslogks</td>
<td>Information about active locks</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysloginroles</td>
<td>One row for each server login that possesses a system-defined role</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>syslogins</td>
<td>One row for each valid user account</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sysmessages</td>
<td>One row for each system error or warning</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysprocesses</td>
<td>Information about server processes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysremotelogs</td>
<td>One row for each remote user</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>syssrvroles</td>
<td>One row for each server-wide role</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysservers</td>
<td>One row for each remote server</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysusages</td>
<td>One row for each disk piece allocated to a database</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

#### Tables in the Adaptive Server Enterprise sybsecurity database

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
<th>Data?</th>
<th>Supported by IQ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysaudits</td>
<td>One row for each audit record</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sysauditoptions</td>
<td>One row for each global audit option</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
CHAPTER 9 System Tables

About this chapter
The structure of every Sybase IQ database is described in a number of system tables.

The DUMMY system table is the only system table you are permitted to access directly. For all other system tables, you access their underlying data via their corresponding views. See Chapter 8, “System Views.”

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>System table list</td>
<td>591</td>
</tr>
<tr>
<td>DUMMY system table</td>
<td>594</td>
</tr>
</tbody>
</table>

System table list

Table 9-1 lists all Sybase IQ system tables.
### Table 9-1: List of system tables

<table>
<thead>
<tr>
<th>System table</th>
<th>Internal use only?</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUMMY</td>
<td>No</td>
</tr>
<tr>
<td>ISYSARTICLE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSARTICLECOL</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSATTRIBUTE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSATTRIBUTENAME</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSCAPABILITY</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSCHECK</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSCOLPERM</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSCOLSTAT</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSCONSTRAINT</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSDBFILE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSDBSPACE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSDBSPACEPERM</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSDependency</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSDOMAIN</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSEVENT</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSEXTERNENV</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSEXTERNENVOBJECT</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSEXTERNLOGIN</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSFKEY</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSGROUP</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSHISTORY</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIDX</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIDXCOL</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQBACKUPHISTORY</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQBACKUPHISTORYDETAIL</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQDBFILE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQDBSPACE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQIDX</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQINFO</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQJOINIDX</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQJOINIXCOLUMN</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQMPXLOGINPOLICYOPTION</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQMPXSERVER</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQPARTITIONCOLUMN</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSIQTAB</td>
<td>Yes</td>
</tr>
<tr>
<td>System table</td>
<td>Internal use only?</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>ISYSIQTABCOL</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSJAR</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSJARCOMPONENT</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSJAVACLASS</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSLOGINMAP</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSLOGINPOLICY</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSLOGINPOLICYOPTION</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSMVOPTION</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSMVOPTIONNAME</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSOBJECT</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSOPTION</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSOPTSTAT</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSPARTITION</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSPARTITIONKEY</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSPARTITIONSCHEME</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSPHYSIDX</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSPROCEDURE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSPROCPARM</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSPROCPERM</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSPROXYTAB</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSPUBLICATION</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSREMARK</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSREMOTEOPTION</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSREMOTEOPTIONTYPE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSCREDIT</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSREMOTEUSER</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSCHEDULE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSERVER</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSOURCE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSQLSERVERTYPE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSUBPARTITIONKEY</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSUBSCRIPTION</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSYNC</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSYNCPROFILE</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSSYNCSRIPTION</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSTAB</td>
<td>Yes</td>
</tr>
<tr>
<td>ISYSTABCOL</td>
<td>Yes</td>
</tr>
</tbody>
</table>
The DUMMY table is provided as a table that always has exactly one row. This can be useful for extracting information from the database, as in the following example that gets the current user ID and the current date from the database.

```
SELECT USER, today(*) FROM SYS.DUMMY
```

The DUMMY table is a SQL Anywhere system table. For detailed information, including a link to its SQL Anywhere system view see “DUMMY system table” in the SQL Anywhere documentation at SQL Anywhere 11.0.1 > SQL Anywhere Server - SQL Reference > System Objects > Tables > System tables.
About this appendix

The information in this appendix is intended to simplify migration to Sybase IQ from other Sybase databases, and to serve as a guide for creating Sybase IQ applications that are compatible with Adaptive Server Enterprise or SQL Anywhere. Beginning with an overview of Transact-SQL, it compares these databases in several areas to be aware of when you are moving to Sybase IQ:

- Architecture
- Data types
- Data definition language
- Data manipulation language
- Stored procedure language

Compatibility features are addressed in each new version of Sybase IQ. This appendix compares Sybase IQ 15.1 with Adaptive Server Enterprise 15.0.3 (and earlier releases), and SQL Anywhere 11.0.1.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>An overview of Transact-SQL support</td>
<td>596</td>
</tr>
<tr>
<td>Adaptive Server architectures</td>
<td>597</td>
</tr>
<tr>
<td>Data types</td>
<td>601</td>
</tr>
<tr>
<td>Data definition language</td>
<td>607</td>
</tr>
<tr>
<td>Data manipulation language</td>
<td>616</td>
</tr>
<tr>
<td>Transact-SQL procedure language overview</td>
<td>626</td>
</tr>
<tr>
<td>Automatic translation of stored procedures</td>
<td>630</td>
</tr>
<tr>
<td>Returning result sets from Transact-SQL procedures</td>
<td>630</td>
</tr>
<tr>
<td>Variables in Transact-SQL procedures</td>
<td>631</td>
</tr>
<tr>
<td>Error handling in Transact-SQL procedures</td>
<td>632</td>
</tr>
<tr>
<td>SQL Anywhere and Sybase IQ</td>
<td>634</td>
</tr>
<tr>
<td>Adaptive Server Enterprise and Sybase IQ</td>
<td>638</td>
</tr>
</tbody>
</table>
An overview of Transact-SQL support

Compatibility information elsewhere in this book

You can find additional compatibility information in the following chapters:

- Chapter 3, “SQL Data Types.” See compatibility information for each data type; also see Data type conversions on page 89.
- Reference: Statements and Options. See the compatibility information in each command.

A note about SQL Anywhere

Sybase IQ is an extension of SQL Anywhere. In most cases, SQL syntax, functions, options, utilities, procedures, and other features are common to both products. There are, however, important differences. Do not assume that features described in SQL Anywhere documentation are supported for Sybase IQ.

The Sybase IQ documentation set calls many, but not all differences. Sybase IQ documentation always supersedes the SQL Anywhere documentation. Except for topics where the Sybase IQ documentation refers you to SQL Anywhere documentation, always refer to the documentation listed as “Documentation for Sybase IQ” in “About This Book,” immediately after the Table of Contents in each Sybase IQ book.

An overview of Transact-SQL support

Sybase IQ, like SQL Anywhere, supports a large subset of Transact-SQL, which is the dialect of SQL supported by Sybase Adaptive Server Enterprise.

The goal of Transact-SQL support in Sybase IQ is to provide application portability. Many applications, stored procedures, and batch files can be written for use with both Adaptive Server Enterprise and Sybase IQ databases.

The aim is to write applications to work with both Adaptive Server Enterprise and Sybase IQ. Existing Adaptive Server Enterprise applications generally require some changes to run on SQL Anywhere or Sybase IQ databases.

Transact-SQL support in Sybase IQ takes the following form:
• Most SQL statements are compatible between Sybase IQ and Adaptive Server Enterprise.

• For some statements, particularly in the procedure language used in procedures and batches, a separate Transact-SQL statement is supported along with the syntax supported in earlier versions of Sybase IQ. For these statements, SQL Anywhere and Sybase IQ support two dialects of SQL. In this appendix, we name those dialects Transact-SQL and Watcom-SQL.

• A procedure or batch is executed in either the Transact-SQL or Watcom-SQL dialect. You must use control statements from one dialect only throughout the batch or procedure. For example, each dialect has different flow control statements.

**Similarities and differences**

Sybase IQ supports a high percentage of Transact-SQL language elements, functions, and statements for working with existing data. Further, Sybase IQ supports a very high percentage of the Transact-SQL stored procedure language (CREATE PROCEDURE syntax, control statements, and so on), and many, but not all, aspects of Transact-SQL data definition language statements.

There are design differences in the architectural and configuration facilities supported by each product. Device management, user management, and maintenance tasks such as backups tend to be system-specific. Even here, however, Sybase IQ provides Transact-SQL system tables as views, where the tables that are not meaningful in Sybase IQ have no rows. Also, Sybase IQ provides a set of system procedures for some of the more common administrative tasks.

**Adaptive Server architectures**

Adaptive Server Enterprise, SQL Anywhere, and Sybase IQ are complementary products, with architectures designed to suit their distinct purposes. Sybase IQ is a high-performance decision-support server designed specifically for data warehousing and analytic processing. SQL Anywhere works well as a workgroup or departmental server requiring little administration, and as a personal database. Adaptive Server Enterprise works well as an enterprise-level server for large databases, with a focus on transaction processing.
This section describes architectural differences among the three products. It also describes the Adaptive Server Enterprise-like tools that Sybase IQ and SQL Anywhere include for compatible database management.

**Servers and databases**

The relationship between servers and databases is different in Adaptive Server Enterprise from Sybase IQ and SQL Anywhere.

In Adaptive Server Enterprise, each database exists inside a server, and each server can contain several databases. Users can have login rights to the server, and can connect to the server. They can then connect to any of the databases on that server, provided that they have permissions. System-wide system tables, held in a master database, contain information common to all databases on the server.

In Sybase IQ, there is nothing equivalent to the Adaptive Server Enterprise master database. Instead, each database is an independent entity, containing all of its system tables. Users can have connection rights to a database, rather than to the server. When a user connects, he or she connects to an individual database. There is no system-wide set of system tables maintained at a master database level. Each Sybase IQ database server can dynamically start and stop a database, to which users can maintain independent connections. Sybase strongly recommends that you run only one Sybase IQ database per server.

SQL Anywhere and Sybase IQ provide tools in their Transact-SQL support and Open Server support to allow some tasks to be carried out in a manner similar to Adaptive Server Enterprise. There are differences, however, in exactly how these tools are implemented.

See the *System Administration Guide: Volume 2*, Chapter 3, “Sybase IQ as a Data Server,” for details on how to use isql to connect to a specific database on a server with multiple databases.

**Space allocation and device management**

All three products use different models for managing devices and allocating disk space initially and later, reflecting the different uses for the products. For example:
• In Adaptive Server Enterprise, you allocate space in database devices initially using DISK INIT and then create a database on one or more database devices. You can add more space using ALTER DATABASE or automatically, using thresholds.

• In Sybase IQ, you initially allocate space by listing raw devices in the CREATE DATABASE statement. You can add more space manually using CREATE DBSPACE. Although you cannot add space automatically, you can create events to warn the DBA before space is actually needed. Sybase IQ can also use file system space. Sybase IQ does not support Transact-SQL DISK statements, such as DISK INIT, DISK MIRROR, DISK REFIT, DISK REINIT, DISK REMIRROR, and DISK UNMIRROR.

• SQL Anywhere is similar to Sybase IQ, except that the initial CREATE DATABASE statement takes a single file system file instead of a list of raw devices. SQL Anywhere also lets you initialize its databases using a command utility dbinit, which Sybase IQ does not support.

For information on disk management, see the System Administration Guide: Volume 1.

System tables, catalog store, and IQ store

An IQ database is a joint data store consisting of:

• The catalog store includes system tables and stored procedures, and resides in a set of tables that are compatible with SQL Anywhere.

• The permanent IQ store is the set of Sybase IQ tables. Table data is stored in indexes.

• The temporary store consists of a set of temporary tables which the database server uses for sorting and other temporary processing.

Catalog distinctions and compatibility features include:

• SQL Anywhere and Sybase IQ use a different schema from Adaptive Server Enterprise for the catalog (tables, columns, and so on).
• SQL Anywhere and Sybase IQ provide compatibility views that mimic relevant parts of the Adaptive Server Enterprise system tables, although there are performance implications when using them. For a list and individual descriptions, see Chapter 8, “System Views” and Chapter 9, “System Tables.” For a complete list of SQL Anywhere compatibility views, see *SQL Anywhere Server – SQL Reference*.

• In Adaptive Server Enterprise, the database owner (user ID *dbo*) owns the catalog objects.

• In SQL Anywhere and Sybase IQ, the system owner (user ID *SYS*) owns the catalog objects.

**Note**  A *dbo* user ID owns the Adaptive Server Enterprise-compatible system views provided by Sybase IQ. The *dba* user ID owns a small number of Sybase IQ system tables, used for Sybase IQ user and multiplex administration.

### Administrative roles

Adaptive Server Enterprise has a more elaborate set of administrative roles than either SQL Anywhere or Sybase IQ. In Adaptive Server Enterprise, there is a set of distinct roles, although more than one login account on an Adaptive Server Enterprise can be granted any role, and one account can possess more than one role.

In Adaptive Server Enterprise, distinct roles include:

• **System administrator**  Responsible for general administrative tasks unrelated to specific applications; can access any database object.

• **System security officer**  Responsible for security-sensitive tasks in Adaptive Server Enterprise, but has no special permissions on database objects.

• **Database owner**  Has full permissions on objects inside the database he or she owns, can add users to a database and grant other users the permission to create objects and execute commands within the database.

• **Data definition statements**  Permissions can be granted to users for specific data definition statements, such as *CREATE TABLE* or *CREATE VIEW*, enabling the user to create database objects.
• **Object owner** Each database object has an owner who may grant permissions to other users to access the object. The owner of an object automatically has all permissions on the object.

In SQL Anywhere and Sybase IQ, the following database-wide permissions have administrative roles:

• **Database Administrator (DBA authority)** Has, like the Adaptive Server Enterprise Database Owner, full permissions on all objects inside the database (other than objects owned by SYS) and can grant other users the permission to create objects and execute commands within the database. The default database administrator is user ID DBA.

• **RESOURCE permission** Allows a user to create any kind of object within a database. This is in place of the Adaptive Server Enterprise scheme of granting permissions on individual CREATE statements.

• **Object owner** Sybase IQ has object owners in the same way Adaptive Server Enterprise does. The owner of an object automatically has all permissions on the object, including the right to grant permissions.

For seamless access to data held in both Adaptive Server Enterprise and Sybase IQ, create user IDs with appropriate permissions in the database (RESOURCE in Sybase IQ, or permission on individual CREATE statements in Adaptive Server Enterprise) and create objects from that user ID. If you use the same user ID in each environment, object names and qualifiers can be identical in the two databases, providing compatible access.

---

**Data types**

This section discusses compatibility information for data types. For details of Sybase IQ data types, see Chapter 3, “SQL Data Types.”

---

**Note** Data types that are not included in this section are currently supported by all three products.
**Data types**

**Bit data type**

All three products support the BIT data type, with these differences:

- SQL Anywhere permits only 0 or 1.
- Adaptive Server Enterprise and Sybase IQ implicitly convert integral data types to BIT. Nonzero values are stored as 1 (TRUE).

**Character data types**

All three products permit CHAR and VARCHAR data, but each product treats these types differently.

- SQL Anywhere treats all strings as VARCHAR, even in a blank-padded database.
- Adaptive Server Enterprise and Sybase IQ differentiate between CHAR (fixed-length) and VARCHAR (variable-length) data.
  
  Adaptive Server Enterprise trims trailing blank spaces from VARCHAR values. Sybase IQ trims trailing blanks from VARCHAR values depending on the form of the data and the operation. See "Character data types" in Chapter 3, “SQL Data Types.”

When inserting into CHAR or VARCHAR:

- SQL Anywhere permits inserting integral data types into CHAR or VARCHAR (implicit conversion).
- Adaptive Server Enterprise and Sybase IQ require explicit conversion.

The maximum size of a column is determined as follows:

- Adaptive Server Enterprise CHAR and VARCHAR depend on the logical page size, which can be 2K, 4K, 8K, and 16K. For example:
  
  - 2K page size allows a column as large as a single row, about 1962 bytes.
  - 4K page size allows a column as large as about 4010 bytes.
- SQL Anywhere supports up to 32K-1 with CHAR and VARCHAR, and up to 2GB with LONG VARCHAR.
- SQL Anywhere supports the name LONG VARCHAR and its synonym TEXT, while Adaptive Server Enterprise supports only the name TEXT, not the name LONG VARCHAR.
• Sybase IQ supports CHAR and VARCHAR up to 32K-1 bytes.
Sybase IQ also supports up to 512TB (with an IQ page size of 128KB) and 2PB (with an IQ page size of 512KB) with LONG VARCHAR. For information on the LONG VARCHAR data type in Sybase IQ, see *Large Objects Management in Sybase IQ*.

• Adaptive Server Enterprise supports NCHAR, NVARCHAR, UNICHAR, UNIVARCHAR data types. N is for multibyte character sets; UNI is for single-byte character sets.

• SQL Anywhere and Sybase IQ support Unicode in the CHAR and VARCHAR data types, rather than as a separate data type.

• For compatibility between Sybase IQ and Adaptive Server Enterprise, always specify a length for character data types.

### Binary data types

The following table summarizes binary data type support.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Adaptive Server Enterprise</th>
<th>SQL Anywhere</th>
<th>Sybase IQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>&lt; page size</td>
<td>32KB - 1</td>
<td>255</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>&lt; page size</td>
<td>32KB - 1</td>
<td>32KB - 1</td>
</tr>
<tr>
<td>LONG BINARY*</td>
<td>not supported</td>
<td>2GB - 1</td>
<td>512TB (IQ page size 128KB) 2PB (IQ page size 512KB)</td>
</tr>
<tr>
<td>IMAGE</td>
<td>2GB</td>
<td>2GB - 1</td>
<td>use LONG BINARY*</td>
</tr>
</tbody>
</table>

*For information on the LONG BINARY data type in Sybase IQ, see *Large Objects Management in Sybase IQ*. This feature requires a separate license.

Adaptive Server Enterprise and SQL Anywhere display binary data differently when projected:

• Sybase IQ supports both Adaptive Server Enterprise and SQL Anywhere display formats.

• If ‘123’ is entered in a BINARY field the SQL Anywhere display format is by bytes, as ‘123’; the Adaptive Server Enterprise display format is by nibbles, as ‘0x616263’.
Date, time, datetime, and timestamp data types

Although all three products support some form of date and time data, there are some differences.

- SQL Anywhere and Sybase IQ support the 4-byte date and time data types.
- Adaptive Server Enterprise supports an 8-byte datetime type, and timestamp as a user-defined data type (domain) implemented as binary (8).
- SQL Anywhere and Sybase IQ support an 8-byte timestamp type, and an 8-byte datetime domain implemented as timestamp. The millisecond precision of the Anywhere/Sybase IQ datetime data type differs from that of Adaptive Server Enterprise.

Display formats for dates have different defaults:

- Adaptive Server Enterprise defaults to displaying dates in the format “MMM-DD-YYYY” but can be changed by setting an option.
- SQL Anywhere and Sybase IQ default to the ISO “YYYY-MM-DD” format but can be changed by setting an option.

Time conversions are as follows:

- Adaptive Server Enterprise varies the way it converts time stored in a string to an internal time, depending on whether the fraction part of the second was delimited by a colon or a period.
- SQL Anywhere and Sybase IQ convert times in the same way, regardless of the delimiter.

When you insert a time into a DATETIME column:

- Adaptive Server Enterprise and Sybase IQ default to supplying 1st January 1900.
- SQL Anywhere defaults to supplying the current date.

TIME and DATETIME values retrieved from an Adaptive Server Enterprise database change when inserted into a Sybase IQ table with a DATETIME column using INSERT….LOCATION. The INSERT….LOCATION statement uses Open Client, which has a DATETIME precision of 1/300 of a second.

For example, assume that the following value is stored in a table column in an Adaptive Server Enterprise database:

\[ 2004-11-08 \ 10:37:22.823 \]
When you retrieve and store it in a Sybase IQ table using INSERT...LOCATION, the value becomes:

```
2004-11-08 10:37:22.823333
```

A DATETIME or TIME value retrieved from an Adaptive Server Enterprise database using INSERT...LOCATION can have a different value due to the datetime precision of Open Client.

For example, the DATETIME value in the Adaptive Server Enterprise database is '2004-11-08 10:37:22.823' as retrieved using INSERT...LOCATION is ‘2004-11-08 10:37:22.823333’.

**Numeric data types**

Adaptive Server Enterprise, SQL Anywhere, and Sybase IQ have different default precision and scale:

- In Adaptive Server Enterprise, the default is precision 18 scale 0.
- In SQL Anywhere, the default is precision 30 scale 6.
- In Sybase IQ, the default is precision 126 scale 38. Because these defaults are too large for TDS and for some client tools, always specify a precision and scale for Sybase IQ exact numeric types.

**Approximate numeric data types**

Adaptive Server Enterprise differs from SQL Anywhere and Sybase IQ in how the FLOAT(precision) data type is interpreted: that is, when to create a 4-byte data type and when to create an 8-byte data type.

REAL values are four bytes, DOUBLE values are eight bytes. Turning on the FLOAT_AS_DOUBLE option makes the FLOAT keyword behave like Adaptive Server Enterprise's FLOAT keyword when a precision is not specified. Since Adaptive Server Enterprise treats its own FLOAT values as DOUBLE, enabling the option makes Sybase IQ to treat FLOAT values in the same way Adaptive Server Enterprise treats FLOAT values.

Turning off the FLOAT_AS_DOUBLE option makes the FLOAT keyword behave like a Sybase IQ and SQL Anywhere four byte REAL value.
For Adaptive Server Enterprise, the precision specified in `FLOAT(precision)` means decimal precision. For SQL Anywhere and Sybase IQ, precision is an integer expression that specifies the number of bits used in the mantissa of the floating point number.

For more information on the Sybase IQ FLOAT data type, see "Numeric data types" in Chapter 3, “SQL Data Types.”

**Text data type**

Support for TEXT data is currently implemented as follows:

- Adaptive Server Enterprise and SQL Anywhere support up to 2GB with LONG VARBINARY and TEXT.
- Sybase IQ supports up to 32KB - 1 with VARCHAR. Sybase IQ also supports up to 512TB (with an IQ page size of 128KB) and 2PB (with an IQ page size of 512KB) with LONG VARCHAR. For information on the LONG VARCHAR data type in Sybase IQ, see Large Objects Management in Sybase IQ.

**Image data type**

Support for IMAGE data is currently implemented as follows:

- Adaptive Server Enterprise and SQL Anywhere support up to 2GB with IMAGE.
- Sybase IQ supports up to 512TB (with an IQ page size of 128KB) and 2PB (with an IQ page size of 512KB) with LONG BINARY. For information on the LONG BINARY data type in Sybase IQ, see Large Objects Management in Sybase IQ.

**Java data types**

Adaptive Server Enterprise allows Java data types in the database. SQL Anywhere and Sybase IQ do not.
Data definition language

This section discusses compatibility information for creating database objects. See also “System tables, catalog store, and IQ store” on page 599 and “Space allocation and device management” on page 598 for related information.

Creating a Transact-SQL compatible database

This section describes choices you must make when creating or rebuilding a database.

Here are the basic steps you need to take to create a Transact-SQL compatible database. The remainder of the section describes which options you need to set.

❖ Creating a Transact-SQL compatible database from Sybase Central

1. One page of the Create Database wizard is named Default Database Attributes.


❖ Creating a Transact-SQL compatible database using the CREATE DATABASE statement

• Type the following statement, for example, in Interactive SQL:

```
CREATE DATABASE 'db-name.db'
CASE RESPECT BLANK PADDING ON
```

Case-sensitivity

Case-sensitivity in databases refers to:

• **Data**  The case-sensitivity of the data is reflected in indexes, in the results of queries, and so on.

• **Identifiers**  Identifiers include table names, column names, user IDs, and so on.

• **Passwords**  Case-sensitivity of passwords is treated differently from other identifiers.
You decide the case-sensitivity of Sybase IQ data in comparisons when you create the database. By default, Sybase IQ databases are case-sensitive in comparisons, although data is always held in the case in which you enter it.

Adaptive Server Enterprise sensitivity to case depends on the sort order installed on the Adaptive Server Enterprise system. You can change case-sensitivity for single-byte character sets by reconfiguring the Adaptive Server Enterprise sort order.

Sybase IQ does not support case-sensitive identifiers. In Adaptive Server Enterprise, the case-sensitivity of identifiers follows the case-sensitivity of the data.

In Adaptive Server Enterprise, user-defined data type names are case-sensitive. In Sybase IQ, they are case-insensitive.

In Sybase IQ and SQL Anywhere, all passwords in newly-created databases are case-sensitive, regardless of the case-sensitivity of the database. The default user ID is DBA and the password for this user is lowercase sql.

When you rebuild an existing database, Sybase IQ and SQL Anywhere determine the case-sensitivity of the password as follows:

- If the database was originally entered in a case-insensitive database, the password remains case-insensitive.
- If the password was originally entered in a case-sensitive database, uppercase and mixed-case passwords remain case-sensitive. If the password was entered in all lowercase, then the password becomes case-insensitive.
- Changes to both existing passwords and new passwords are case-sensitive.

In Adaptive Server Enterprise, the case-sensitivity of user IDs and passwords follows the case-sensitivity of the server.

Each database object must have a unique name within a certain name space. Outside this name space, duplicate names are allowed. Some database objects occupy different name spaces in Adaptive Server Enterprise as compared to SQL Anywhere and Sybase IQ.
Table name uniqueness

Table name uniqueness requirements apply within a database:

- For Sybase IQ and SQL Anywhere, table names must be unique within a database for a given owner. For example, both user1 and user2 can create a table called employee; uniqueness is provided by the fully qualified names, user1.employee and user2.employee.

- For Adaptive Server Enterprise, table names must be unique within the database and to the owner.

Index name uniqueness

Index name uniqueness requirements apply within a table. In all three products, indexes are owned by the owner of the table on which they are created. Index names must be unique on a given table, but any two tables can have an index of the same name, even for the same owner. For example, in all three products, tables t1 and t2 can have indexes of the same name, whether they are owned by the same or different users.

Renaming indexes and foreign keys

Sybase IQ allows you to rename explicitly created indexes, foreign key role names of indexes, and foreign keys, using the ALTER INDEX statement. Adaptive Server Anywhere allows you to rename indexes, foreign key role names, and foreign keys, using the ALTER INDEX statement. Adaptive Server Enterprise does not allow you to rename these objects.

CREATE TABLE statement

When creating tables for compatibility, be aware of the following items.

NULL in columns

For compatible treatment of NULL:

- SQL Anywhere and Sybase IQ assume that columns can be null unless NOT NULL is stated in the column definition. You can change this behavior by setting the database option ALLOW_NULLS_BY_DEFAULT to the Transact-SQL compatible setting of OFF.

- SQL Anywhere assumes that BIT columns only cannot be NULL.

- Adaptive Server Enterprise assumes that columns cannot be null unless NULL is stated.

Check constraints

Sybase IQ enforces check constraints on base, global temporary, and local temporary tables, and on user-defined data types. Users can log check integrity constraint violations and specify the number of violations that can occur before a LOAD statement rolls back.
Sybase IQ does not allow the creation of a check constraint that it cannot evaluate, such as those composed of user-defined functions, proxy tables, or non-Sybase IQ tables. Constraints that cannot be evaluated are detected the first time the table on which the check constraint is defined is used in a LOAD, INSERT, or UPDATE statement. Sybase IQ does not allow check constraints containing:

- Subqueries
- Expressions specifying a host language parameter, a SQL parameter, or a column as the target for a data value
- Set functions
- Invocations of nondeterministic functions or functions that modify data

Adaptive Server Enterprise and SQL Anywhere enforce CHECK constraints. SQL Anywhere allows subqueries in check constraints.

Sybase IQ supports user-defined data types that allow constraints to be encapsulated in the data type definition.

Sybase IQ enforces referential integrity as described in Chapter 9, “Ensuring Data Integrity” in the System Administration Guide: Volume 1.

Actions for enforcing integrity are supported as follows:

- SQL Anywhere supports all ANSI actions: SET NULL, CASCADE, DEFAULT, RESTRICT.
- Adaptive Server Enterprise supports two of these actions: SET NULL, DEFAULT.

**Note** You can achieve CASCADE in Adaptive Server Enterprise by using triggers instead of referential integrity.

- Sybase IQ supports the RESTRICT action only.
- Sybase IQ does not support NOT NULL FOREIGN KEY.
- Sybase IQ has the restriction that a column cannot be both a candidate key and a foreign key at the same time.

Default value support differs as follows:

- Adaptive Server Enterprise and SQL Anywhere support specifying a default value for a column.
- Only SQL Anywhere supports DEFAULT UTC TIMESTAMP.
APPENDIX A  Compatibility with Other Sybase Databases

Identity columns

- Sybase IQ supports specifying a default value for a column, except for the special values DEFAULT UTC TIMESTAMP and DEFAULT CURRENT UTC TIMESTAMP. Sybase IQ also ignores settings for the DEFAULT_TIMESTAMP_INCREMENT database option.

- Sybase IQ supports IDENTITY or DEFAULT AUTOINCREMENT as a default value. Sybase IQ supports identity columns of any numeric type with any precision and scale 0, and the column can be NULL. Sybase IQ identity columns must be positive and are limited by the range of the data type. Sybase IQ supports a single identity column per table, and requires database option IDENTITY_INSERT set to a table name for explicit inserts and updates. To drop a table with an IDENTITY column, you cannot have IDENTITY_INSERT set to that table. The table can contain data when adding an identity column. Tables derived using SELECT INTO do not have Identity/Autoincrement columns. Sybase IQ views cannot contain IDENTITY/DEFAULT AUTOINCREMENT columns.

- SQL Anywhere supports the AUTOINCREMENT default value. SQL Anywhere supports identity columns of any numeric type with any allowable scale and precision. The identity column value can be positive, negative, or zero, limited by the range of the data type. SQL Anywhere supports any number of identity columns per table, and does not require identity_insert for explicit inserts, drops, and updates. The table must be empty when adding identity columns. SQL Anywhere identity columns can be altered to be nonidentity columns, and vice versa. You can add or drop AUTOINCREMENT columns from SQL Anywhere views.

- Adaptive Server Enterprise supports a single identity column per table. ASE identity columns are restricted to only numeric data type scale 0, maximum precision 38. They must be positive, are limited by the range of the data type, and cannot be null. Adaptive Server Enterprise requires identity_insert for explicit inserts and drops, but not for updates to the identity column. The table can contain data when you add an identity column. ASE users cannot explicitly set the next value chosen for an identity column. ASE views cannot contain IDENTITY/AUTOINCREMENT columns. When using SELECT INTO under certain conditions, ASE allows Identity/Autoincrement columns in the result table if they were in the table being selected from.

Computed columns

Computed column support differs as follows:

Reference: Building Blocks, Tables, and Procedures 611
Data definition language

- SQL Anywhere supports computed columns that can be indexed.
- Adaptive Server Enterprise and Sybase IQ do not.

Temporary tables
You can create a temporary table by placing a pound sign (#) without an owner specification in front of the table name in a CREATE TABLE statement. These temporary tables are Sybase IQ-declared temporary tables and are available only in the current connection. For information about declared temporary tables in Sybase IQ, see the "DECLARE LOCAL TEMPORARY TABLE statement" in Reference: Statements and Options.

See the "CREATE TABLE statement" in Reference: Statements and Options.

Locating tables
Physical placement of a table is carried out differently in Adaptive Server Enterprise and Sybase IQ. Sybase IQ supports the ON segment-name clause, but segment-name refers to a Sybase IQ dbspace.

CREATE DEFAULT, CREATE RULE, and CREATE DOMAIN statements
Sybase IQ provides an alternative means of incorporating rules:
- Adaptive Server Enterprise supports the Create Default and Create Rule statements to create named defaults.
- SQL Anywhere and Sybase IQ support the CREATE DOMAIN statement to achieve the same objective.

CREATE TRIGGER statement
Support for triggers differs as follows:
- SQL Anywhere supports both row-level and statement-level triggers.
- Adaptive Server Enterprise supports only statement-level triggers.
• Sybase IQ does not support triggers.

**Note** A trigger is effectively a stored procedure that is run automatically either immediately before or immediately after an **INSERT**, **UPDATE**, or **DELETE** as part of the same transaction, that can be used to cause a dependent change (for example, to automatically update the name of an employee’s manager when the employee is moved to a different department). It can also be used to write an audit trail to identify which modifications made which changes to the database, and at what time.

---

**CREATE INDEX statement**

CREATE INDEX syntax differs slightly among the three products:

• Adaptive Server Enterprise and SQL Anywhere support clustered or nonclustered indexes, using the following syntax:

```
CREATE [UNIQUE] [CLUSTERED] INDEX name
ON table (column,...)
ON dbspace
```

Adaptive Server Enterprise also allows the **NONCLUSTERED** keyword, but for both products the default is **NONCLUSTERED**.

• Adaptive Server Enterprise CREATE INDEX statements work in SQL Anywhere because SQL Anywhere allows, but ignores, the keywords **FILLFACTOR**, **IGNORE_DUP_KEY**, **SORTED_DATA**, **IGNORE_DUP_ROW**, and **ALLOW_DUP_ROW**.

• SQL Anywhere CREATE INDEX syntax supports the **VIRTUAL** keyword for use by its Index Consultant, but not for actual query executions.

• Sybase IQ supports seven specialized index types: LF, HG, HNG, DATE, TIME, DTTM, and WD. Sybase IQ also supports a CMP index on the relationship between two columns of identical data type, precision, and scale. Sybase IQ defaults to creating an HG index unless the index type is specified in the CREATE INDEX statement:

```
CREATE [UNIQUE] [type] INDEX name
```
Data definition language

ON table (column,...)

**Note** Sybase IQ also supports CREATE JOIN INDEX, which lets you create a prejoined index on a certain set of columns that are joined consistently and frequently in queries.

See Chapter 6, “Using Sybase IQ Indexes” in the *System Administration Guide: Volume 1*.

Users, groups, and permissions

There are some differences between the Adaptive Server Enterprise and SQL Anywhere and Sybase IQ models of users and groups.

**How users connect**

In Adaptive Server Enterprise, users connect to a server, and each user requires a login ID and password to the server as well as a user ID for each database they want to access on that server.

SQL Anywhere and Sybase IQ users do not require a server login ID. All SQL Anywhere and Sybase IQ users receive a user ID and password for a database.

**User groups**

All three products support user groups, so you can grant permissions to many users at one time. However, there are differences in the specifics of groups:

- Adaptive Server Enterprise allows each user to be a member of only one group.
- SQL Anywhere and Sybase IQ allow users to be members of multiple groups, and group hierarchies are allowed.

All three products have a public group, for defining default permissions. Every user automatically becomes a member of the public group.

**Database object permissions**

GRANT and REVOKE statements for granting permissions on individual database objects are very similar in all three products.

- All three products allow SELECT, INSERT, DELETE, UPDATE, and REFERENCES permissions on database tables and views, and UPDATE permissions on selected columns of database tables.

For example, the following statement is valid in all three products:

```
GRANT INSERT, DELETE
ON TITLES
```
TO MARY, SALES

This statement grants permission to use the INSERT and DELETE statements on the TITLES table to user MARY and to the SALES group.

- All three products allow EXECUTE permissions to be granted on stored procedures.
- Adaptive Server Enterprise also supports GRANT and REVOKE on additional items:
  - Objects: columns within tables, columns within views, and stored procedures
  - User abilities: CREATE DATABASE, CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW
- SQL Anywhere and Sybase IQ require a user to have RESOURCE authority to create database objects. (A closely corresponding Adaptive Server Enterprise permission is GRANT ALL, used by a Database Owner.)
- All three products support the WITH GRANT OPTION clause, allowing the recipient of permissions to grant them in turn, although Sybase IQ does not permit WITH GRANT OPTION to be used on a GRANT EXECUTE statement.

Database-wide permissions

Adaptive Server Enterprise uses a different model for database-wide user permissions.

- SQL Anywhere and Sybase IQ employ DBA permissions to allow a user full authority within a database.
- The System Administrator in Adaptive Server Enterprise enjoys this permission for all databases on a server. However, DBA authority on a Sybase IQ database is different from the permissions of an Adaptive Server Enterprise Database Owner, who must use the Adaptive Server Enterprise SETUSER statement to gain permissions on objects owned by other users.

Adding users

Adaptive Server Enterprise requires a two-step process to add a user: sp_addlogin followed by sp_add_user.

SQL Anywhere and Sybase IQ add users in a single step.

Sybase IQ Login Management stored procedures, although not required to add or drop users, allow DBAs to add or drop Sybase IQ user accounts. When Sybase IQ User Administration is enabled, these Sybase IQ user accounts let DBAs control user connections and password expirations.
Data manipulation language


Although SQL Anywhere and Sybase IQ allow Adaptive Server Enterprise system procedures for managing users and groups, the exact syntax and function of these procedures differs in some cases. For more information, see Chapter 7, “System Procedures,” including “Adaptive Server Enterprise system procedures” on page 522.

Load formats

Load format support in the three products is as follows:

- Sybase IQ handles ASCII, BINARY, and BCP load formats.
- SQL Anywhere, in addition to ASCII and BINARY, also lets you import dBase, Excel, FoxPro, and Lotus file formats.
- Adaptive Server Enterprise handles ASCII and BINARY load formats through BCP.

Note The syntax of the Sybase IQ and SQL Anywhere LOAD statement is based on BCP and designed to offer exactly the same functionality.

Setting options for Transact-SQL compatibility

Set Sybase IQ database options using the SET OPTION statement. See “Transact-SQL compatibility options” in “Database Options,” Reference: Statements and Options, for a list of option settings required for compatible behavior.

Data manipulation language

This section provides some general guidelines for writing portable queries, then discusses specific query requirements.
General guidelines for writing portable SQL

When writing SQL for use on more than one database management system, make your SQL statements as explicit as possible. Even if more than one server supports a given SQL statement, it might be a mistake to assume that default behavior is the same on each system. General guidelines applicable to writing compatible SQL include:

• Spell out all of the available options, rather than using default behavior.

• Use parentheses to make the order of execution within statements explicit, rather than assuming identical default order of precedence for operators.

• Use the Transact-SQL convention of an @ sign preceding variable names for Adaptive Server Enterprise portability.

• Declare variables and cursors in procedures and batches immediately following a BEGIN statement. Sybase IQ requires this, although Adaptive Server Enterprise allows declarations to be made anywhere in a procedure or batch.

• Do not use reserved words from either Adaptive Server Enterprise or Sybase IQ as identifiers in your databases.

Writing compatible queries

There are two criteria for writing a query that runs on both Sybase IQ and Adaptive Server Enterprise databases:

• The data types, expressions, and search conditions in the query must be compatible.

• The syntax of the SELECT statement itself must be compatible.

Sybase IQ supports the following subset of the Transact-SQL SELECT statement.

Syntax

```
SELECT [ ALL | DISTINCT ] select-list
  [... INTO #temporary-table-name ]
  [... FROM table-spec,
    ... table-spec, ... ]
  [... WHERE search-condition ]
  [... GROUP BY column-name, ... ]
  [... HAVING search-condition ]
```
**Data manipulation language**

... | [ ORDER BY expression [ ASC | DESC ], ... ] |  
| [ ORDER BY integer [ ASC | DESC ], ... ] |

**Parameters**

select-list:

{ table-name. * }...
{ * }...
{ expression }...
{ alias-name = expression }...
{ expression as identifier }...
{ expression as T_string }...

table-spec:

{ [ owner. ]table-name...
[ [ AS ] correlation-name ] }
...

alias-name:

identifier | 'string' | "string"

For a full description of the SELECT statement, see "SELECT statement" in *Reference: Statements and Options*.

The sections that follow provide details on several items to be aware of when writing compatible queries.

**Subqueries**

Sybase IQ currently provides support for subqueries that is somewhat different from that provided by Adaptive Server Enterprise and SQL Anywhere. Adaptive Server Enterprise and SQL Anywhere support subqueries in the ON clause; Sybase IQ does not currently support this.

UNION in subqueries is supported as follows:

- SQL Anywhere supports UNION in both correlated and uncorrelated subqueries.
- Sybase IQ supports UNION only in uncorrelated subqueries.
- Adaptive Server Enterprise does not support UNION in any subqueries.

SQL Anywhere supports subqueries in many additional places that a scalar value might appear in the grammar. Adaptive Server Enterprise and Sybase IQ follow the ANSI standard as to where subqueries can be specified.
GROUP BY clause

GROUP BY ALL support is as follows:

- Adaptive Server Enterprise supports GROUP BY ALL, which returns all possible groups including those eliminated by the WHERE clause and HAVING clause. These have the NULL value for all aggregates.
- SQL Anywhere does not support the GROUP BY ALL Transact-SQL extension.

ROLLUP and CUBE in the GROUP BY clause are supported as follows:

- Sybase IQ and SQL Anywhere support ROLLUP and CUBE in the GROUP BY clause.
- Adaptive Server Enterprise does not currently support ROLLUP and CUBE.

Adaptive Server Enterprise supports projecting nongrouped columns in the SELECT clause. This is known as extended group by semantics and returns a set of values. Sybase IQ supports and SQL Anywhere do not support extended group by semantics. Only SQL Anywhere supports the List() aggregate to return a list of values.

For information about using GROUP BY with OLAP functions, see Chapter 2, “Using OLAP” in the System Administration Guide: Volume 2.

COMPUTE clause

COMPUTE clause support is as follows:

- Adaptive Server Enterprise supports the Transact-SQL COMPUTE clause.
- SQL Anywhere and Sybase IQ do not support the Transact-SQL COMPUTE clause since it is not in the ANSI standard and this functionality is provided by most third-party front-end tools.

WHERE clause

The WHERE clause differs in support for the Contains() predicate, and treatment of trailing white space in the Like() predicate.
Data manipulation language

- Sybase IQ supports the Contains() predicate for word searches in character data (similar to Contains in MS SQL Server and Verity). Sybase IQ uses WORD indexes to optimize these, if possible.
- SQL Anywhere and Adaptive Server Enterprise do not support Contains().

Joins

This section discusses support for Transact-SQL outer joins and ANSI joins.

Transact-SQL outer joins

Supported syntax for outer joins can be summarized as follows:
- Adaptive Server Enterprise fully supports *= and =* Transact-SQL syntax for outer joins.
- SQL Anywhere and Sybase IQ support Transact-SQL outer joins, but reject some complex Transact-SQL outer joins that are potentially ambiguous.
- Sybase IQ does not support chained (nested) Transact-SQL outer joins. Use ANSI syntax for this type of multiple outer join.

Note Transact-SQL outer join syntax is deprecated in SQL Anywhere and Sybase IQ.

ANSI joins

Support for ANSI join syntax can be summarized as follows:
- Sybase IQ does not currently support subqueries in the ON clause.
- Adaptive Server Enterprise and SQL Anywhere support subqueries in the ON clause.

Full outer join support is as follows:
- SQL Anywhere and Sybase IQ support FULL OUTER JOIN.
- Adaptive Server Enterprise does not support FULL OUTER JOIN.
More information on outer joins

For detailed information on Transact-SQL outer joins, including ANSI syntax alternatives, see the white paper “Semantics and Compatibility of Transact-SQL Outer Joins,” from MySybase at http://www.sybase.com/support/. Although written for SQL Anywhere, the information in this document also applies to Sybase IQ.

Null comparisons

Adaptive Server Enterprise has Transact-SQL extensions that permit predicates to compare the null value. For example, `{col} = Null` means `{col} Is Null.

SQL Anywhere and Sybase IQ use ANSI semantics for null comparisons unless the ANSINULL option is set to OFF, in which case such comparisons are Adaptive Server Enterprise-compatible.

Note SQL Anywhere 8.0 and later adds support for the TDS_EMPTY_STRING_AS_NULL to offer Adaptive Server Enterprise compatibility in mapping empty strings to the null value.

Zero-length strings

Zero-length strings are treated as follows:

- Adaptive Server Enterprise treats zero-length strings as the null value. Adaptive Server Enterprise users store a single space for blank strings.
- SQL Anywhere and Sybase IQ follow ANSI semantics for zero-length strings, that is, a zero-length string is a real value; it is not null.

HOLDLOCK, SHARED, and FOR BROWSE

Support for this syntax is as follows:

- Adaptive Server Enterprise supports HOLDLOCK, SHARED, and FOR BROWSE syntax.
Data manipulation language

- SQL Anywhere supports HOLDLOCK but does not support SHARED or FOR BROWSE.
- Sybase IQ does not support these keywords.

SQL functions

Sybase IQ supports most of the same functions as SQL Anywhere and Adaptive Server Enterprise, with these differences:

- Adaptive Server Enterprise supports the USING CHARACTERS | USING BYTES syntax in PatIndex(); SQL Anywhere and Sybase IQ do not.
- Adaptive Server Enterprise supports the Reverse() function; SQL Anywhere and Sybase IQ do not.
- Adaptive Server Enterprise supports Len() as alternative syntax for Length(); SQL Anywhere does not support this alternative.
- Adaptive Server Enterprise supports the Square() and Str_Replace() Microsoft compatibility functions; SQL Anywhere does not.
- Sybase IQ supports Str_Replace().
- Adaptive Server Enterprise and SQL Anywhere support TSEQUAL() to compare two timestamps for modification time; Sybase IQ does not support TSEQUAL(). (TSEQUAL is not relevant in the Sybase IQ table-level versioning model.)
- Sybase IQ supports ROWID(); Adaptive Server Enterprise and SQL Anywhere do not.
- SQL Anywhere and Sybase IQ support Cast() in addition to Adaptive Server Enterprise’s Convert() for data type conversions.

**Note**  Cast() is the ANSI-compliant name.

- SQL Anywhere and Sybase IQ support Lcase() and Ucase() as synonyms of Lower() and Upper(); Adaptive Server Enterprise does not.
- SQL Anywhere and Sybase IQ support the Locate() string function; Adaptive Server Enterprise does not.
• SQL Anywhere supports the `IsDate()` and `IsNumeric()` function to test the ability to convert a string to the respective data type; Adaptive Server Enterprise does not. Sybase IQ supports `IsDate()`. You can use `IsNumeric` in Sybase IQ, but CIS functional compensation performance considerations apply.

• SQL Anywhere supports the `NEWID`, `STRTOUID`, and `UUIDTOSTR` functions; Adaptive Server Enterprise does not. These are native functions in Sybase IQ, so CIS functional compensation performance considerations do not apply.

**Note** Some SQL functions, including `SOUNDEX` and `DIFFERENCE` string functions, and some date functions operate differently in Sybase IQ and SQL Anywhere. The Sybase IQ database option `ASE_FUNCTION_BEHAVIOR` specifies that output of some of the Sybase IQ data type conversion functions, including `HEXTOINT` and `INTTOHEX`, is consistent with the output of Adaptive Server Enterprise functions.

### OLAP functions

Sybase IQ currently supports these OLAP functions:

• `Corr()`
• `Covar_Pop()`
• `Covar_Samp()`
• `Cume_Dist`
• `Dense_Rank()`
• `Exp_Weighted_Avg`
• `First_Value`
• `Last_Value`
• `Median`
• `Ntile()`
• `Percent_Rank()`
• `Percentile_Cont()`
• `Percentile_Disc()`
Data manipulation language

- Rank()
- Regr_Avgx()
- Regr_Avgy()
- Regr_Intercept()
- Regr_R2
- Regr_Slope()
- Regr_Sxx()
- Regr_Sxy()
- Regr_Syy()
- StdDev()
- Stddev_Pop
- Stddev_Samp
- Var_Pop
- Var_Samp
- Variance()
- Weighted_Avg

SQL Anywhere supports all of the Sybase IQ OLAP functions. Currently, Adaptive Server Enterprise does not support OLAP functions. CIS functional compensation does not support OLAP functions.

**Note** Support for OLAP functions is a rapidly evolving area of Sybase product development. See Chapter 4, “SQL Functions.” Also see Chapter 2, “Using OLAP” in the *System Administration Guide: Volume 2.*

System functions

SQL Anywhere and Sybase IQ do not support the following Adaptive Server Enterprise system functions:
- `curunreservedpgs()` – number of pages free on a dbspace.
- `data_pgs()` – number of pages used by a table or index.
• host_id() – UNIX pid of the server process.
• hos_name() – name of the machine on which the server is running.
• lct_admin() – manages the “last chance threshold” for the Transaction manager.
• reserved_pgs() – number of pages allocated to a table or index.
• rowcnt() – number of rows in the specified table.
• valid_name() – whether a name would be a valid name if used, for example, for a table.
• valid_user() – returns TRUE if that user has connect permissions.
• ptn_data_pgs() – number of data pages in a partition.
• index_colorder() – returns the column order in an index.

User-defined functions

User-defined function (UDF) support varies as follows:
• SQL Anywhere supports UDFs in SQL, Java, and C.
• Adaptive Server Enterprise supports UDFs written only in Java.
• Sybase IQ offers support for UDFs via CIS query decomposition, but there are performance implications.

Arithmetic expressions on dates

SQL Anywhere and Sybase IQ interpret arithmetic expressions on dates as shorthand notation for various date functions. Adaptive Server Enterprise does not.
• Date +/- integer is equivalent to Dateadd().
• Date – date is equivalent to Datediff().
• Date + time creates a timestamp from the two.
SELECT INTO

There are differences in the types of tables permitted in a statement like the following:

```sql
select into table1 from table2
```

- Adaptive Server Enterprise permits `table1` to be permanent, temporary or a proxy table. Adaptive Server Enterprise also supports `SELECT INTO EXISTING TABLE`.

- SQL Anywhere and Sybase IQ permit `table1` to be a permanent or a temporary table. A permanent table is created only when you select into `table` and specify more than one column. `SELECT INTO #table`, without an owner specification, always creates a temporary table, regardless of the number of columns specified. `SELECT INTO table` with just one column selects into a host variable.

Updatable views

Adaptive Server Enterprise and SQL Anywhere are more liberal than ANSI permits on the view definitions that are updatable when the `WITH CHECK` option is not requested.

SQL Anywhere offers the `ANSI_UPDATE_CONSTRAINTS` option to control whether updates are restricted to those supported by SQL92, or a more liberal set of rules.

Sybase IQ permits `UPDATE` only on single-table views that can be flattened. Sybase IQ does not support `WITH CHECK`.

FROM clause in UPDATE and DELETE

All three products support the `FROM` clause with multiple tables in `UPDATE` and `DELETE`.

Transact-SQL procedure language overview

The stored procedure language is the part of SQL used in stored procedures and batches.
SQL Anywhere and Sybase IQ support a large part of the Transact-SQL stored procedure language in addition to the Watcom-SQL dialect based on SQL92.

**Transact-SQL stored procedure overview**

Because it is based on the ISO/ANSI draft standard, the SQL Anywhere and Sybase IQ stored procedure language differs from the Transact-SQL dialect in many ways. Many of the concepts and features are similar, but the syntax is different. SQL Anywhere and Sybase IQ support for Transact-SQL takes advantage of the similar concepts by providing automatic translation between dialects. However, you must write a procedure exclusively in one of the two dialects, not in a mixture of the two.

There are a variety of aspects to SQL Anywhere and Sybase IQ support for Transact-SQL stored procedures, including:

- Passing parameters
- Returning result sets
- Returning status information
- Providing default values for parameters
- Control statements
- Error handling

**Transact-SQL batch overview**

In Transact-SQL, a batch is a set of SQL statements submitted together and executed as a group, one after the other. Batches can be stored in command files. The ISQL utility in SQL Anywhere and Sybase IQ and the *isql* utility in Adaptive Server Enterprise provide similar capabilities for executing batches interactively.

The control statements used in procedures can also be used in batches. SQL Anywhere and Sybase IQ support the use of control statements in batches and the Transact-SQL-like use of nondelimited groups of statements terminated with a GO statement to signify the end of a batch.
For batches stored in command files, SQL Anywhere and Sybase IQ support the use of parameters in command files. Adaptive Server Enterprise does not support parameters.

For information on parameters, see "PARAMETERS statement [DBISQL]" in Reference: Statements and Options.

SQL statements in procedures and batches

Some SQL statements supported by Sybase IQ are part of one dialect, but not the other. You cannot mix the two dialects within a procedure or batch. This means that:

- You can include Transact-SQL-only statements with statements that are part of both dialects in a batch or procedure.
- You can include statements not supported by Adaptive Server Enterprise with statements that are supported by both servers in a batch or procedure.
- You cannot include Transact-SQL-only statements with Sybase IQ—only statements in a batch or procedure.

SQL statements not separated by semicolons are part of a Transact-SQL procedure or batch. See Reference: Statements and Options for details of individual statements.

Expression subqueries in IF statements

Adaptive Server Enterprise and SQL Anywhere support comparisons between a variable and a scalar value returned by an expression subquery. For example:

```sql
create procedure testIf ()
begin
    declare var4 int;
    set var4 = 10;
    if var4 = (select MIN (a_i1) from a) then set var4 = 100;
end if;
end;
```
CASE statement

Permitted usage of the CASE statement differs in Sybase IQ and SQL Anywhere.

The CASE statement is not supported in Adaptive Server Enterprise, which supports case expressions only.

For a detailed comparison of case expression support in Sybase IQ and Adaptive Server Enterprise, see “Expressions” on page 23.

Row-level cursor operations

All three products support the use of cursors with UPDATE and DELETE as follows:

```sql
UPDATE WHERE CURRENT OF {cursor}
DELETE WHERE CURRENT OF {cursor}
```

In Sybase IQ, updatable cursors are asensitive only, for one table only, and chained only. Updatable hold cursors are not permitted. Updatable cursors in Sybase IQ get a table lock.

Print command

The effect of the PRINT command depends on the client:

- Adaptive Server Enterprise PRINT always sends a message to the client.
- In SQL Anywhere and Sybase IQ, PRINT sends a message to the client for Open Client and JDBC connections.
- Adaptive Server Enterprise stored procedures that rely on PRINT work in Sybase IQ using DBISQL.

Note   Sybase IQ users might prefer dbisql with JDBC, rather than the iAdaptive Server Anywhere JDBC driver (formerly called the JDBC-ODBC bridge).
Automatic translation of stored procedures

In addition to supporting Transact-SQL alternative syntax, SQL Anywhere and Sybase IQ provide aids for translating statements between the Watcom-SQL and Transact-SQL dialects. Functions returning information about SQL statements and enabling automatic translation of SQL statements include:

- **SQLDialect(statement)** Returns Watcom-SQL or Transact-SQL.
- **WatcomSQL(statement)** Returns the Watcom-SQL syntax for the statement.
- **TransactSQL(statement)** Returns the Transact-SQL syntax for the statement.

These are functions and thus can be accessed using a SELECT statement from ISQL. For example, the following statement returns the value Watcom-SQL:

```sql
SELECT SqlDialect('select * from Employees')
```

Returning result sets from Transact-SQL procedures

SQL Anywhere and Sybase IQ use a RESULT clause to specify returned result sets. In Transact-SQL procedures, column names or alias names of the first query are returned to the calling environment.

Example of Transact-SQL procedure

The following Transact-SQL procedure illustrates how Transact-SQL stored procedures return result sets:

```sql
CREATE PROCEDURE showdept (@deptname varchar(30)) AS
    SELECT Employees.Surname, Employees.GivenName
    FROM Departments, Employees
    WHERE Departments.DepartmentName = @deptname
    AND Departments.DepartmentID = Employees.DepartmentID
```

Example of Watcom-SQL procedure

The following is the corresponding SQL Anywhere or Sybase IQ procedure:

```sql
CREATE PROCEDURE showdept(in deptname varchar(30)) RESULT ( lastname char(20), firstname char(20))
BEGIN
```
SELECT Employees.Surname, Employees.GivenName
FROM Departments, Employees
WHERE Departments.DepartmentName = deptname
AND Departments.DepartmentID = Employee.DepartmentID
END

Multiple result sets

There are minor differences in the way the three Sybase client tools present multiple results to the client:

- ISQL displays all results in a single stream.
- DBISQL presents each result set on a separate tab. You must enable this functionality in the Option menu. Make it a permanent change, then restart or reconnect to DBISQL.
- DBISQLC provides the RESUME command to display each successive result set.

For more information about procedures and results, see Chapter 1, “Using Procedures and Batches” in the System Administration Guide: Volume 2.

Variables in Transact-SQL procedures

SQL Anywhere and Sybase IQ use the SET statement to assign values to variables in a procedure. In Transact-SQL, values are assigned using the SELECT statement with an empty table list. The following simple procedure illustrates how the Transact-SQL syntax works:

```
CREATE PROCEDURE multiply
    @mult1 int,
    @mult2 int,
    @result int output
AS
SELECT @result = @mult1 * @mult2
```

This procedure can be called as follows:

```
CREATE VARIABLE @product int
go
EXECUTE multiply 5, 6, @product OUTPUT
go
```

The variable @product has a value of 30 after the procedure executes.
There are some differences in order and persistence of variable declarations:

- In Adaptive Server Enterprise, you can declare variables anywhere in the body of a stored procedure. Variables persist for the duration of the procedure.

- In SQL Anywhere and Sybase IQ, you must declare variables at the beginning of a compound statement (that is, immediately after `BEGIN` in a `BEGIN...END` pair). Variables persist only for the duration of the compound statement.

For more information on using the `SELECT` statement to assign variables, see “Writing compatible queries” on page 617. For more information on using the `SET` statement to assign variables, see "SET statement [ESQL]" in Reference: Statements and Options.

---

**Error handling in Transact-SQL procedures**

Default procedure error handling is different in the Watcom-SQL and Transact-SQL dialects. By default, Watcom-SQL dialect procedures exit when they encounter an error, returning SQLSTATE and SQLCODE values to the calling environment.

You can build explicit error handling into Watcom-SQL stored procedures using the `EXCEPTION` statement, or you can instruct the procedure to continue execution at the next statement when it encounters an error, using the `ON EXCEPTION RESUME` statement.

When a Transact-SQL dialect procedure encounters an error, execution continues at the following statement. The global variable `@@error` holds the error status of the most recently executed statement. You can check this variable following a statement to force return from a procedure. For example, the following statement causes an exit if an error occurs:

```
IF @@error != 0 RETURN
```

When the procedure completes execution, a return value indicates the success or failure of the procedure. This return status is an integer, and can be accessed as follows:

```
DECLARE @status INT
EXECUTE @status = proc_sample
IF @status = 0
```
PRINT 'procedure succeeded'
ELSE
  PRINT 'procedure failed'

Table A-2 describes the built-in procedure return values and their meanings:

Table A-2: Built-in procedure return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Procedure executed without error</td>
</tr>
<tr>
<td>-1</td>
<td>Missing object</td>
</tr>
<tr>
<td>-2</td>
<td>Data type error</td>
</tr>
<tr>
<td>-3</td>
<td>Process was chosen as deadlock victim</td>
</tr>
<tr>
<td>-4</td>
<td>Permission error</td>
</tr>
<tr>
<td>-5</td>
<td>Syntax error</td>
</tr>
<tr>
<td>-6</td>
<td>Miscellaneous user error</td>
</tr>
<tr>
<td>-7</td>
<td>Resource error, such as out of space</td>
</tr>
<tr>
<td>-8</td>
<td>Nonfatal internal problem</td>
</tr>
<tr>
<td>-9</td>
<td>System limit was reached</td>
</tr>
<tr>
<td>-10</td>
<td>Fatal internal inconsistency</td>
</tr>
<tr>
<td>-11</td>
<td>Fatal internal inconsistency</td>
</tr>
<tr>
<td>-12</td>
<td>Table or index is corrupt</td>
</tr>
<tr>
<td>-13</td>
<td>Database is corrupt</td>
</tr>
<tr>
<td>-14</td>
<td>Hardware error</td>
</tr>
</tbody>
</table>

The RETURN statement can be used to return other integers, with their own user-defined meanings.

**Using the RAISERROR statement in procedures**

The RAISERROR statement is a Transact-SQL statement for generating user-defined errors. It has a similar function to the SIGNAL statement.

For a description of the RAISERROR statement, see "RAISERROR statement [T-SQL]" in Reference: Statements and Options.

By itself, RAISERROR does not cause an exit from the procedure, but it can be combined with a RETURN statement or a test of the @@error global variable to control execution following a user-defined error.
If you set the ON_TSQL_ERROR database option to CONTINUE, RAISERROR no longer signals an execution-ending error. Instead, the procedure completes and stores the RAISERROR status code and message, and returns the most recent RAISERROR. If the procedure causing the RAISERROR was called from another procedure, RAISERROR returns after the outermost calling procedure terminates.

You lose intermediate RAISERROR statuses and codes when the procedure terminates. If, at return time, an error occurs along with RAISERROR, the error information is returned and you lose the RAISERROR information. The application can query intermediate RAISERROR statuses by examining @@error global variable at different execution points.

**Transact-SQL-like error handling in the Watcom-SQL dialect**

You can make a Watcom-SQL dialect procedure handle errors in a Transact-SQL-like manner by supplying the ON EXCEPTION RESUME clause to the CREATE PROCEDURE statement:

```sql
CREATE PROCEDURE sample_proc()
ON EXCEPTION RESUME
BEGIN
    ...
END
```

The presence of an ON EXCEPTION RESUME clause prevents explicit exception handling code from being executed, so avoid using these two clauses together.

**SQL Anywhere and Sybase IQ**

The preceding sections, while focused on compatibility with Transact-SQL, also clarify many of the distinctions between Sybase IQ and SQL Anywhere.

This section points out other differences and shared functionality between Sybase IQ and SQL Anywhere.
For additional information, always refer to the Sybase IQ documentation set when using the product. Refer to the SQL Anywhere documentation set when using SQL Anywhere, or when the Sybase IQ documentation refers to SQL Anywhere documentation for specific functionality only.

**Server and database startup and administration**

Note the following differences in starting and managing databases and servers:

- Sybase IQ uses the server startup command `start_iq`, instead of the SQL Anywhere network server startup command.
- Sybase IQ does not support personal servers.
- Sybase IQ supports many SQL Anywhere server command line options, but not all. Other server options are supported for Sybase IQ but not for SQL Anywhere.
- Sybase IQ provides the `stop_iq` utility to shut down servers.
- Clauses permitted in the `BACKUP` and `RESTORE` statements differ in Sybase IQ and SQL Anywhere.
- SQL Remote is supported in Sybase IQ only for multiplex operations.

Sybase IQ supports many SQL Anywhere database administration utilities, but not all:

- The following SQL Anywhere utilities are *not* supported by Sybase IQ: `backup`, `compression`, `console`, `initialization`, `license`, `log transfer`, `log translation`, `rebuild`, `spawn`, some transaction log options (`-g`, `-il`, `-ir`, `-n`, `-x`, `-z`), `uncompression`, `unload`, `upgrade`, and `write file`.
- Sybase IQ supports the SQL Anywhere validation utility only on the catalog store. To validate the IQ store, use `sp_iqcheckdb`.

**Database options**

Some SQL Anywhere database options are not supported by Sybase IQ, including `DEFAULT_TIMESTAMP_INCREMENT`.

Some database options apply only to the catalog store, including: `FOR_XML_NULL_TREATMENT`, `ISOLATION_LEVEL`, `PREFETCH`, `PRECISION`, and `SCALE`. 
Options with differences in behavior, default, or allowed values include DELAYED_COMMITS, TIME_FORMAT, TIMESTAMP_FORMAT.

Sybase IQ also includes many options that SQL Anywhere does not support. See Chapter 2, “Database Options.” in Reference: Statements and Options.

**Data definition language (DDL)**

In addition to the DDL differences discussed previously:

- In a DELETE/DROP or PRIMARY KEY clause of an ALTER TABLE statement, Sybase IQ takes the RESTRICT action (reports an error if there are associated foreign keys). SQL Anywhere always takes the CASCADE action.
- Similarly, DROP TABLE statement reports an error in Sybase IQ if there are associated foreign-key constraints.
- Sybase IQ does not support these DDL statements: CREATE COMPRESSED DATABASE, CREATE TRIGGER, SETUSER.
- Sybase IQ supports referential integrity at the statement level, rather than the transaction-level integrity that SQL Anywhere supports with the CHECK ON COMMIT clause of the CREATE TABLE statement.
- A Sybase IQ table cannot have a foreign key that references a SQL Anywhere (or catalog) table, and a SQL Anywhere table cannot have a foreign key that references a Sybase IQ table.
- In a Sybase IQ database, publications can only be created on SQL Anywhere tables.
- In CREATE DATABASE, the defaults for case-sensitivity and collation differ. The defaults for Sybase IQ are CASE RESPECT and the ISO_BINENG collation; for SQL Anywhere, the defaults are CASE IGNORE, and collation inferred from the language and character set of the operating system.
Data manipulation language (DML)

- Sybase IQ does not support these DML and procedural statements: EXPLAIN, GET DATA, INPUT, PREPARE TO COMMIT, PUT, READTEXT, ROLLBACK TRIGGER, SYSTEM, UNLOAD TABLE, VALIDATE TABLE.

**Note** A set of extraction options perform a role similar to UNLOAD TABLE; for details, see the section “Data extraction options” in Chapter 7, “Moving Data In and Out of Databases” in the System Administration Guide: Volume 1.

- Sybase IQ supports the INSERT...LOCATION syntax; SQL Anywhere does not.
- LOAD TABLE options differ in Sybase IQ and SQL Anywhere.
- OPEN statement in Sybase IQ does not support BLOCK and ISOLATION LEVEL clauses.
- Sybase IQ does not support triggers.
- Use of transactions, isolation levels, checkpoints, and automatically generated COMMITS, as well as cursor support, is different in Sybase IQ and SQL Anywhere. See Chapter 10, “Transactions and Versioning” in the System Administration Guide: Volume 1.
- When you SELECT from a stored procedure in Sybase IQ, CIS functional compensation performance considerations apply. See “Conditions that cause processing by SQL Anywhere” in Performance and Tuning Guide.
- Sybase IQ ignores the database name qualifier in fully qualified names in Adaptive Server Enterprise SELECT statements, such as a FROM clause with `<database name>.<owner>.<table name>`. For example, Sybase IQ interprets the query `SELECT * FROM XXX..TEST` as `SELECT * FROM TEST`.

Stored procedures

See “SQL Anywhere supported procedures” on page 525.

Obsolete procedures and views are removed from databases created in versions earlier than Sybase IQ 12.6 ESD #2 when you run the ALTER DATABASE UPGRADE command using Sybase IQ 12.6 ESD #2 or later.
If you do not run ALTER DATABASE UPGRADE, the procedures remain in the database.

**Adaptive Server Enterprise and Sybase IQ**

This section points out other differences and shared functionality between Sybase IQ and Adaptive Server Enterprise.

For additional information, always refer to the Sybase IQ documentation set when using the product. Refer to the Adaptive Server Enterprise documentation set when using Adaptive Server Enterprise, or when the Sybase IQ documentation refers to Adaptive Server Enterprise documentation for specific functionality only.

**Stored procedures**

Sybase IQ no longer supports these Adaptive Server stored procedures:
- `sp_addserver`
- `sp_configure`
- `sp_estspace`
- `sp_help`
- `sp_helpuser`
- `sp_who`

Sybase IQ no longer supports the following catalog procedures:
- `sp_column_privileges`
- `sp_databases`
- `sp_datatype_info`
- `sp_server_info`

Obsolete procedures and views are removed from databases created prior to Sybase IQ 15.1 ESD #2 when you run the ALTER DATABASE UPGRADE command using Sybase IQ 15.1 ESD #2 or higher.
If you do not run ALTER DATABASE UPGRADE, the procedures remain in the database.

System views

Sybase IQ no longer supports these Adaptive Server Enterprise views:
- sysalternates
- sysaudits
- sysauditoptions
- sysconstraints
- syscharsets
- sysconfigures
- syscurconfigs
- sysdatabases
- sysdepends
- sysdevices
- sysengines
- syskeys
- syslanguages
- syslocks
- sylogs
- sysloginroles
- sysmessages
- sysprocedures
- sysprocesses
- sysprotects
- sysreferences
- sysremotelogins
- sysroles
Column name differences

The column name used in the Adaptive Server Enterprise view SYSTYPES is “allownulls”. The column name used in the Sybase IQ view SYSTYPES is “allowsnulls”.

- syssegments
- sysservers
- syssrvroles
- systhresholds
- sysusages
Index

A
ABS function 128
absolute value 128
ACOS function 128
Adaptive Server Enterprise compatibility 595
advice clearing 418
displaying 418
storing 418
aggregate functions 102
AVG 131
COUNT 146
MAX 201
MIN 203
STDDEV 257
STDDEV_POP 258
STDDEV_SAMP 259
SUM 265
VAR_POP 331
VAR_SAMP 332
VARIANCE 333
ALL conditions 39
allocation map resetting 363
alphabetic characters defined 21
analytic functions
DENSE_RANK 163
NTILE 212
PERCENT_RANK 218
PERCENTILE_CONT 220
PERCENTILE_DISC 222
RANK 229
analytical functions 104
AND conditions 45
ANY conditions 39
apostrophe
in strings 22
approximate numeric data types compatibility 605
arc-cosine 128
architectures
Adaptive Server 597
arc-sine 129
arc-tangent 130
arc-tangent ratio 130
ARGN function 129
argument selection 129
arithmetic expressions 26
on dates 625
ASCII function 129
ASCII value 129, 135
ASIN function 129
ATAN function 130
ATAN2 function 130
auditing
adding comments 492
audits disabling 500
enabling 498
automatic joins and foreign keys 344
average 131
AVG function 131
Backslashes
not allowed in SQL identifiers 21
backup history file location 8
backup operations summary 358
backups during checkpoint 492
Index

during low activity 492
batches
  Transact-SQL overview 627
  writing 627
BETWEEN conditions 40
BIGINTTOHEX function 132
binary data
  compatibility 603
  BINARY data type 77
  binary load format
    data file 94
    LOAD TABLE 94
BIT data type
  compatibility 602
  Transact-SQL 82
  bit length 132
  BIT_LENGTH function 132
bitwise operators 27
blanks
  trimming trailing 71, 602
brackets
  database objects 21
  SQL identifiers 21
buffer cache
  monitoring with sp_iqsysmon 466
byte length 216
BYTE_LENGTH function 133

C
  cache
    flushing 501
    main 506
    temporary 510
  CASE expression 29
    NULLIF function 213
case sensitivity
  and pattern matching 41
  comparison conditions 34
data 608
databases 607
identifiers 608
passwords 608
Transact-SQL compatibility 607
user IDs 608
user-defined data types 607, 608
  CAST function 89, 133
catalog
  Adaptive Server Enterprise compatibility 599
  system tables 591
catalog store
  IQ 599
    monitoring with 466
    validating 511
CEIL function 134
CEILING function 135
chained outer joins 620
CHAR data type
  about 69
  CHAR function 135
  CHAR_LENGTH function 136
character data
  compatibility 602
  CHARACTER data type
    about 70
  character sets
    specifying 11
  CHARACTER VARYING data type
    about 70
  trimming trailing blanks 71, 602
  CHARINDEX function 136
CHECK conditions
  Transact-SQL 609
  check constraints 609
  enforced 609
  Transact-SQL compatibility 609
CHECKPOINT statement
  backup during checkpoint 492
  clauses
    ON EXCEPTION RESUME 634
  COALESCE function 137
code pages
  and data storage 70
  COL_LENGTH function 138
  COL_NAME function 138
collation
  SORTKEY function 251
column default
  not supported 610
column length 138
column name 138
columns
  and user-defined data types 88
  naming 25
  SYSCOLUMNS system view 537
command-line options
  overriding 504
comments
  comment indicators 65
comparing dates and times 85
comparisons
  about 33
compatibility
  Adaptive Server Enterprise 595
  approximate numeric data 605
  referential integrity constraints 610
COMPUTE clause
  Transact-SQL 619
computed columns
  not supported 611
concatenating strings 26
condition hint strings 48
conditions
  user-supplied for queries 46
connecting
  IMSL library 124
connection information
    sp_iqcontext 378
connection property value 139
CONNECTION_PROPERTY function 139
connection-level variables
  about 59
connections
  determining ID number 209
  displaying information about 487
  number of 373
  properties 121
consistency checking
  partitions 360
consistent state
  restoring 454
constants
  in expressions 25
  Transact-SQL 31
CONTAINS conditions 43
CONVERT function 89, 139
date to string conversion 141
integer to date conversion 141
string to date conversion 141
CORR function 142
COS function 143
cosine 143
COT function 143
cotangent 143
COUNT function 146
COVAR_POP function 144
COVAR_SAMP function 145
CPU utilization
  database consistency checker 361
CREATE DEFAULT statement
  unsupported 612
CREATE DOMAIN statement
  Transact-SQL compatibility 612
  using 87
CREATE EXISTING TABLE statement
  proxy tables 513
CREATE INDEX statement
  IQ 613
  Transact-SQL 613
CREATE RULE statement
  unsupported 612
CREATE TABLE statement
  Transact-SQL 609
CREATE TRIGGER
  not supported 612
creating
  data types 87
CUBE operation
  GROUPING function 177
CUME_DIST function 147
CURRENT DATABASE
  special value 53
CURRENT DATE
  default 53
  special value 53
CURRENT PUBLISHER
  default 53
  special value 53
CURRENT TIME
  default 53
  special value 53
CURRENT_TIMESTAMP
Index

default 54
special value 54
CURRENT USER
default 54
special value 54
current user
environment settings 14
cursors
displaying information about 381
row-level in IQ 629
Transact-SQL 629

D
data
case sensitivity 608
data type compatibility
approximate numeric data 605
binary data 603
bit data 602
character data 602
date and time data 604
datetime and time data 605
IMAGE data 606
Java data 606
numeric data 605
TEXT data 606
data type conversion
about 89
BIT to BINARY 91
BIT to CHAR 92
BIT to VARBINARY 91
BIT to VARCHAR 92
CHAR to BIT 92
functions 108
VARCHAR to BIT 92
data type conversion functions 108
BIGINTTOHEX 132
CAST 133
CONVERT 139
HEXTOBIGINT 178
HEXToint 179
INTTOHEX 189
data types
about 69
Adaptive Server Enterprise 601
and compatibility 90
and roundoff errors 75
binary 77
character 69
date and time 82
displaying information about 384, 408
FLOAT 605
IMAGE 78, 603, 606
IQ 601
LONG BINARY 78, 603, 606
numeric 73
SQL Anywhere 601
TEXT 70, 602, 606
UNIQUEIDENTIFIERSTR 69
user-defined 87
database administrator
roles 601
database object
determining ID 215
determining name 216
identifying 21
database options
DATE_ORDER 86
FLOAT AS DOUBLE 605
QUOTED IDENTIFIER 32
database server
overriding command-line options 504
databases
case sensitivity 607
determining ID number 161, 209
determining name 162
properties 122
property value 162
system procedures 351
system tables 591
validating catalog store 511
DATALength function 148
date and time data types
compatibility 604
date and time functions 109
DATE 149
DATEADD 149
DATEDIFF 151
DATECEILING 150
DATEFLOOR 153

644
Sybase IQ
DATEFORMAT 155
DATENAME 156
DATEPART 157
DATEROUND 157
DATETIME 159
DAY 159
DAYNAME 160
DAYS 160
DOW 165
GETDATE 174
getting consistent results 111
HOUR 180
HOURS 180
IQ features 344
MINUTE 204
MINUTES 204
MONTH 206
MONTHNAME 206
MONTHS 206
NOW 211
QUARTER 227
SECOND 248
SECONDS 248
TODAY 267
WEEKS 335
YEAR 339
YEARS 339
YMD 341
DATE data type 83
DATE function 149
DATE_ORDER option 86
DATEADD function 149
DATECEILING function 150
DATEDIFF function 151
DATEDIF function 151
DATEDIF function 151
DATEFLOOR function 153
DATEFORMAT function 155
DATENAME function 156
DATEPART function 157
DATEROUND function 157
dates
arithmetic expressions 625
determining current 211, 267
interpreting strings as dates 86
queries 85
datetime and time data types
compatibility 605
DATETIME function 159
DAY function 159
day of the week (DOW) 165
DAYNAME function 160
DAYS function 160
DB_ID function 161
DB_NAME function 162
DB_PROPERTY function 162
DBCC
database verification 359
output 365
performance 364
time to run 364
dbce
thread usage 361
DBCC_LOG_PROGRESS option 365
dbcc
not supported 598
dbspaces
managing 598
preventing read-write operations 363
DDL
SQL Anywhere 636
DECIMAL data type 73
default values
CURRENT DATABASE 53
CURRENT PUBLISHER 53
CURRENT USER 54
LAST USER 54
not supported 610
TIMESTAMP 56
USER 56
defaults
CURRENT DATE 53
CURRENT PUBLISHER 53
CURRENT TIME 53
CURRENT TIMESTAMP 54
CURRENT USER 54
Transact-SQL 612
defining a window 105
DEGREES function 163
delimiting SQL strings 21
DENSE_RANK function 163
devices
managing 598
DIFFERENCE function 165
Index

directory structure 2
disjunction of subquery predicates 36
disjunction of subqueries 37
DISK statements
unsupported 598
distribution functions 105
DML
SQL Anywhere 637
documentation
SQL Anywhere xxi
domains
about 87
DOUBLE data type 75
double quotes
database objects 21
not allowed in SQL identifiers 21
DOW function 165
dropleaks mode 363
dummy IQ table 102
getting consistent results 111
DUMMY table 594

ERRORMSG function
SQL syntax 166
errors
Transact-SQL 632, 634
estimates
optimizer 47
EVENT_CONDITION function 166
EVENT_CONDITION_NAME function 168
EVENT_PARAMETER function 168
events
displaying information about 404, 408
EVENT_CONDITION function 166
EVENT_CONDITION_NAME function 168
EVENT_PARAMETER function 168
execution phase hints 49
EXISTS conditions 44
EXP function 169
EXP_WEIGHTED_AVG function 170
exponential function 169
expression
converting to timestamp 159
length in bytes 148
expression subqueries
in IF statements 628
expressions 23
CASE 29
Transact-SQL 31

ELSE
IF expression 29
ENDIF
IF expression 29
environment variables
about 5
IQLOGDIR 15 8
IPORT 8
IQTIMEOUT 9
PATH 11
SQLCONNECT 12
SYBASE 12
SYBASE_JRE 13
SYBASE_OCS 13
error handling
IMSL library 125
error logging
IMSL library 126
error messages
ERRORMSG function 166
getting text 166

F
files
location 3
FIRST_VALUE function 171
FLOAT data type 75
FLOAT_AS_DOUBLE option 605
FLOOR function 173
FOR BROWSE syntax
Transact-SQL 621
foreign keys
system views 542, 543
FP indexes
verifying 362
FROM clause 114
UPDATE and DELETE 626
functions 101
ABS function 128
ACOS function 128
Adaptive Server Enterprise system functions 120
aggregate 102
alphabetical list 128
analytical 104
ARGN function 129
ASCII function 129
ASIN function 129
ATAN function 130
ATAN2 function 130
AVG function 131
BIGINTTOHEX function 132
BIT_LENGTH function 132
BYTE_LENGTH function 133
CAST function 133
CEIL function 134
CEILING function 135
CHAR function 135
CHAR_LENGTH function 136
CHARINDEX function 136
COALESCE function 137
COL_LENGTH function 138
COL_NAME function 138
CONNECTION_PROPERTY function 139
consistent results 114
CONVERT function 139
CORR function 142
COS function 143
cot function 143
COUNT function 146
COVAR_POP function 144
COVAR_SAMP function 145
CUME_DIST function 147
data type conversion 108
DATEDIST function 148
date and time 109
DATE function 149
DATEADD function 149
DATECEILING function 150
DATEDIFF function 151
DATEFLOOR function 153
DATEFORMAT function 155
DATENAME function 156
DATEPART function 157
dATEROUND function 157
DATETIME function 159
DAY function 159
DAYNAME function 160
DAYS function 160
DB_ID function 161
DB_NAME function 162
DB_PROPERTY function 162
DEGREES function 163
DENSE_RANK function 163
difference function 165
distribution 105
DOW function 165
ERROR function 165
EVENT CONDITION function 166
EVENT CONDITION_NAME function 168
EVENT PARAMETER function 168
EXP function 169
EXP_WEIGHTED_AVG function 170
FIRST_VALUE function 171
FLOOR function 173
GETDATE function 174
GRAPHICAL PLAN 174
GROUP MEMBER function SQL syntax 177
GROUPING function SQL syntax 177
HEXTOBIGINT function 178
HEXTOINT function 179
HOUR function 180
HOURS function 180
HTML DECODE function 181
HTML ENCODE function 182
HTTP PLAN function 183
HTTP 114
HTTP DECODE function 185
HTTP ENCODE function 185
HTTP HEADER function 186
HTTP_VARIABLE function 187
IFNULL function 188
INDEX COL function 188
INSERTSTR function 189
INTTOHEX function 189
IQ extensions 345
ISDATE function SQL syntax 191
ISNULL function 191
ISNUMERIC function SQL syntax 192
LAST VALUE 193
LCASE function 194
Index
Index

LEFT function 195
LEN function SQL syntax 196
LENGTH function 197
LOCATE function 198
LOG function 199
LOG10 function 200
LOWER function 200
LTRIM function 201
MAX function 201
MEDIAN function 202
MIN function 203
MINUTE function 204
MINUTES function 204
miscellaneous 127
MOD function 205
MONTH function 206
MONTHNAME function 206
MONTHS function 206
NEWID function SQL syntax 208
NEXT_CONNECTION function 209
NEXT_DATABASE function 209
NEXT_HTTP_HEADER function 210
NEXT_HTTP_VARIABLE function 211
NOW function 211
NTILE function 212
NULLIF function 213
NUMBER function 214
numeric 105, 114
OBJECT_ID function 215
OBJECT_NAME function 216
OCTET_LENGTH function 216
PATINDEX function 217
PERCENT_RANK function 218
PERCENTILE_CONT function 220
PERCENTILE_DISC function 222
PI function 224
POWER function 224
PROPERTY function 225
PROPERTY_DESCRIPTION function 225
PROPERTY_NAME function 226
PROPERTY_NUMBER function 227
QUARTER function 227
RADIANS function 228
RAND function 228
RANK function 229
ranking 104
REGR_AVGX function 230
REGR_ASY function 232
REGR_COUNT function 233
REGR_INTERCEPT function 234
REGR_R2 function 235
REGR_SLOPE function 236
REGR_SXX function 237
REGR_SXY function 238
REGR_SYY function 240
REMAINDER function 241
REPEAT function 241
REPLACE function 242
REPLICATE function 243
REVERSE function SQL syntax 244
RIGHT function 245
ROUND function 245
ROWID function 246
RTRIM function 247
SECOND function 248
SECONDS function 248
SIGN function 249
SIMILAR function 250
SIN function 250
SORTKEY function 251
SOUNDEX function 255
SPACE function 256
SQRT function 256
SQUARE function 257
statistical 105
STDDEV function 257
STDDEV_POP function 258
STDDEV_SAMP function 259
STR function 260
STR_REPLACE function SQL syntax 261
string 115
STRING function 263
STRTOUUID function SQL syntax 263
STUFF function 264
SUBSTR function 265
SUBSTRING function 265
SUM function 265
SUSER_ID function 266
SUSER_NAME function 266
TAN function 267
time series 123
today 594
TODAY function 267
Transact-SQL 622
TRIM function 268
TRUNCNUM function 268
TS_ARMA_AR function 269
TS_ARMA_CONST function 272
TS_ARMA_MA function 276
TS_AUTO_UNI_AR function 283
TS_AUTOCORRELATION function 280
TS_BOX_COX_XFORM function 287
TS_DIFFERENCE function 289
TS_ESTIMATE_MISSING function 294
TS_LACK_OF_FIT function 297
TS_LACK_OF_FIT_P function 301
TS_MAX_ARMA_AR function 304
TS_MAX_ARMA_CONST function 308
TS_MAX_ARMA_LIKELIHOOD function 311
TS_MAX_ARMA_MA function 315
TS_OUTLIER_IDENTIFICATION function 318
TS_PARTIAL_AUTOCORRELATION function 323
TS_VWAP function 327
UCASE function 329
UPPER function 329
USER_ID function 330
USER_NAME function 330
user-defined 122
UUIDTOSTR function SQL syntax 331
VAR_POP function 331
VAR_SAMP function 332
VARIANCE function 333
WEEKS function 335
WEIGHTED_AVG function 336
WIDTH_BUCKET function 337
windowing aggregate 104
YEAR function 339
YEARS function 339
YMD function 341
functions, aggregate
GROUPING 177
functions, data type conversion
ISDATE 191
functions, miscellaneous
ERRORMSG 166
ISNUMERIC 192
NEWID 208
functions, string 183, 196, 244, 261
STRTOUUID 263
UUIDTOSTR 331

G
GETDATE function 174
global variables
about 57, 59
compatibility 62
list of 60
globally unique identifiers
SQL syntax for NEWID function 208
GRAPHICAL_PLAN function 174
GROUP BY
compatibility 619
GROUP_MEMBER function
SQL syntax 177
GROUPING function 177
groups
Adaptive Server Enterprise 614
GUIDs
SQL syntax for NEWID function 208
SQL syntax for STRTOUUID function 263
SQL syntax for UUIDTOSTR function 331

H
HEXTOBIGINT function 178
HEXTOINT function 179
ASE_FUNCTION_BEHAVIOR option 179
hints
execution phase 49, 51
index preference 49
HOLDLOCK syntax
Transact-SQL 621
HOUR function 180
HOURS function 180
HTML DECODE function 181
HTML_ENCODE function 182
HTML_PLAN 183
HTML_PLAN function 183
HTTP
setting headers 510
Index

setting options 511
HTTP functions 114
HTML DECODE 181
HTML_ENCODE 182
HTTP DECODE 185
HTTP_ENCODE 185
HTTP_HEADER 186
HTTP_VARIABLE 187
NEXT_HTTP_HEADER 210
NEXT_HTTP_VARIABLE 211
HTTP DECODE function 185
HTTP_ENCODE function 185
HTTP_HEADER function 186
HTTP_VARIABLE function 187
INSERTSTR function 189
installation directory
about 2
INTEGER data type 73
INTTOHEX function 189
ASE_FUNCTION_BEHAVIOR option 190
IQ Agent
port 8
wait time 9
IQ store 599
iq_dummy table 102
IQDIR15 environment variable 7
IQLOGDIR15 environment variable 8
IQMsgMaxSize server option 505
IQMsgNumFiles server option 505
IQPORT environment variable 8
IQTIMESTAMP environment variable
specifying IQ Agent wait time 9
IQTMP15 environment variable 9
IS NULL conditions 44
ISDATE function
SQL syntax 191
ISNULL function 191
ISNUMERIC function
SQL syntax 192

J
Java
user-defined functions 123
Java data types
compatibility 606
Java Runtime Environment
setting 13
join equality conditions 51
join indexes
displaying information about 427
number of tables
queries
number of tables per block 349
join operators
ANSI 620
Transact-SQL 620
joins
   automatic 344
   outer operators 28
   Transact-SQL 620

K
keys
   displaying information about 442
   verifying 362
keywords
   listing 18
   SQL 17

L
languages
   specifying 11
LAST USER
   special value 54
LAST_VALUE function 193
LCASE function 194
LD_LIBRARY_PATH environment variable 10
LEFT function 195
LEN function
   SQL syntax 196
LENGTH function 197
LIBPATH environment variable 10
library
   IMSL error handling 125
   IMSL error logging 126
LIKE conditions 40
literal strings 22, 25
liveness timeout
   database server 504
load formats
   Transact-SQL and SQL Anywhere 616
LOAD TABLE
   BINARY 94
   BINARY FORMAT 94
   binary format data file 94
local machine
   environment settings 14
local variables
   about 57
   LOCATE function 198
locks
   displaying 434
   LOG function 199
   LOG10 function 200
logarithm (base 10) 200
logarithm of a number 199
login management
   sp_expireallpasswords 354
   sp_iqaddlogin 355
   sp_iqcopyloginpolicy 380, 437
login policies
   assigning user to 438
   copying 380, 437
LONG BINARY data type 78, 603, 606
LOWER function 200
LTRIM function 201
LVC cells 364

M
main_cache_memory_mb
   sa_server_option parameter 506
master database
   unsupported 598
mathematical expressions 26
MAX function 201
MEDIAN function 202
memory
   monitoring with sp_iqsysmon 466
message log
   specifying number of archives 505
   specifying size 505
MIN function 203
MINUTE function 204
MINUTES function 204
miscellaneous functions 127
   ARGN 129
   COALESCE 137
   IFNULL 188
   ISNULL 191
   NULLIF 213
   NUMBER 214
   ROWID 246
Index

MOD function 205
MONEY data type 76
monitor
  sp_iqsysmon procedure 466
MONTH function 206
MONTHNAME function 206
MONTHS function 206
MPXServerName column 374
multiplex
  synchronizing query servers 9
system procedures 373

N
name spaces
  indexes 608
nested outer joins 620
NEWID function
  SQL syntax 208
NEXT_CONNECTION function 209
NEXT_DATABASE function 209
NEXT_HTTP_HEADER function 210
NEXT_HTTP_VARIABLE function 211
NOT conditions 45
NOW function 211
NTILE function 212
NULL
  Transact-SQL compatibility 609
null comparisons
  Transact-SQL 621
null value
  about 67
NULLIF function 30, 213
NUMBER function 214
number of connections
  determining 373
numbers 25
NUMERIC 75
numeric data types
  compatibility 605
numeric functions 105, 114
  ABS 128
  ACOS 128
  ASIN 129
  ATAN 130
  ATAN2 130
  CEIL 134
  CEILING 135
  consistent results 114
  COS 143
  COT 143
  DEGREES 163
  EXP 169
  FLOOR 173
  LOG 199
  LOG10 200
  MOD 205
  PI 224
  POWER 224
  RADIANS 228
  RAND 228
  REMAINDER 241
  ROUND 245
  SIGN 249
  SIN 250
  SQRT 256
  SQUARE 257
  TAN 267
  TRUNCNUM 268
  WIDTH_BUCKET 337

O
object
  defining 501
determining ID 215
determining name 216
displaying information about 408
renaming 452
OBJECT_ID function 215
OBJECT_NAME function 216
OCTET_LENGTH function 216
OLAP
  DENSE_RANK function 163
distribution functions 105
  GROUPING function 177
  NTILE function 212
numeric functions 105
  PERCENT_RANK function 218
  PERCENTILE_CONT function 220
Index

PERCENTILE_DISC function  222
RANK function  229
ranking functions  104
statistical functions  105
STDDEV function  257
VARIANCE function  333
window function type  105
window functions  105
window name  105
window specification  105
windows aggregate functions  104
OLAP functions
  compatibility  623
OLAP OVER clause  105
ON EXCEPTION RESUME clause
  Transact-SQL  634
Open Client setting  13
operators
  comparison operators  34
  precedence of  29
optimizer
  estimates  47
  user-defined selectivity  47
options
  DBCC_LOG_PROGRESS  365
  FLOAT_AS_DOUBLE  605
  QUOTED_IDENTIFIER  32
  SQL Anywhere  635
  system views  565, 584
OR keyword  45
outer joins
  and subqueries  25
  chained  620
  nested  620
  operators  28
  Transact-SQL  620
OVER clause  105

P
partitioned tables
  verifying  363
partitions
  consistency checking  360
passwords
  adding or modifying  441
  case sensitivity  608
  expiring  354
  sa_verify_password system procedure  354, 512
PATH environment variable  11
PATINDEX function  217
pattern matching
  about  40
  and collations  41
  limits  41
PERCENT_RANK function  218
percentile
  computing with NTILE function  212
PERCENTILE_CONT function  220
PERCENTILE_DISC function  222
performance
  monitoring  466
  sp_iqshowpsexe connection information  457
  sp_iqsysmon procedure  466
permissions
  Adaptive Server Enterprise  614
  SYSCOLAUTH system view  534
  system views  579
PI function  224
population variance function  331
portable SQL  617
POWER function  224
preference of operators  29
predicates
  about  33
  disjunction of  36
prefetching
  monitoring with sp_iqsysmon  466
primary keys
  displaying information about  442
  generating unique values  208
  generating unique values using UUIDs  208
  UUIDs and GUIDs  208
PRINT command
  Transact-SQL  629
procedure language
  overview  626
procedures
  displaying information about  408, 443
  displaying parameter information  446
  error handling  632, 634
Reference: Building Blocks, Tables, and Procedures  653
Index

return values 633
Transact-SQL 630
Transact-SQL overview 627
translation 630
properties
connection 121
databases 122
description of ID 225
determining name 226
determining number 227
server 121
server level 225
PROPERTY function 225
PROPERTY_DESCRIPTION function 225
PROPERTY_NAME function 226
PROPERTY_NUMBER function 227
publisher
SQL Remote 53

Q
QUARTER function 227
queries
Transact-SQL 617
query servers
synchronizing 9
quitting time
database server 504
quotation marks
database objects 21
SQL identifiers 21
QUOTED_IDENTIFIER option 32
quotes
strings 22

R
RADIANS function 228
RAISERROR statement
ON EXCEPTION RESUME 634
Transact-SQL 633
RAND function 228
RANK function 229
ranking functions 104
referential integrity constraints
CASCADE not supported 610
compatibility 610
registry entries
about 14
REGR_AVGX function 230
REGR_AVGY function 232
REGR_COUNT function 233
REGR_INTERCEPT function 234
REGR_R2 function 235
REGR_SLOPE function 236
REGR_SXX function 237
REGR_SXY function 238
REGR_SYY function 240
REMAINDER function 241
remote servers
capabilities 518
remote tables
columns 513, 514, 515
listing 517
rename objects
sp_iqrename procedure 452
REPEAT function 241
REPLACE function 242
in SELECT INTO statement 26, 141, 189, 195, 196, 200, 201, 241, 243, 244, 245, 248, 268, 329
REPLICATE function 243
request_level_debugging
about 504
request-level logging
about 504
enabling from Interactive SQL 508
reserved words 18
listing 18
resetclocks
sp_iqcheckdb option 362
restore operations
consistent state 454
result sets
Transact-SQL 630
return values
procedures 633
REVERSE function
SQL syntax 244
RIGHT function 245
roles
<table>
<thead>
<tr>
<th>Reference: Building Blocks, Tables, and Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Adaptive Server Enterprise</strong> 600</td>
</tr>
<tr>
<td><strong>ROLLUP operation</strong></td>
</tr>
<tr>
<td>- <strong>GROUPING function</strong> 177</td>
</tr>
<tr>
<td><strong>ROUND function</strong> 245</td>
</tr>
<tr>
<td><strong>ROWID function</strong> 246</td>
</tr>
<tr>
<td>rows</td>
</tr>
<tr>
<td>- counting 146</td>
</tr>
<tr>
<td><strong>RTRIM function</strong> 247</td>
</tr>
<tr>
<td><strong>rules</strong></td>
</tr>
<tr>
<td>Transact-SQL 612</td>
</tr>
</tbody>
</table>

**S**

sa_audit_string system procedure 492
sa_checkpoint_execute system procedure 492
sa_conn_activity system procedure
  syntax 493
sa_conn_info system procedure 494
sa_conn_properties system procedure 495
sa_db_info system procedure 496
sa_db_properties system procedure 497
sa_dependent_views system procedure 353
sa_disable_auditing_type system procedure 500
sa_enable_auditing_type system procedure 498
sa_eng_properties system procedure 498
sa_flush_cache system procedure 501
sa_make_object system procedure 501
sa_rowgenerator system procedure
  syntax 502
sa_server_option system procedure 504
sa_set_http_header system procedure 510
sa_set_http_option system procedure 511
sa_table_page_usage system procedure 500
sa_validate system procedure
  syntax 511
sa_verify_password system procedure 354
SACHARSET environment variable 11
SALANG environment variable 11
sample variance function 332
search conditions
  about 33
  ALL or ANY conditions 39
  BETWEEN conditions 40
  comparison conditions 34
  CONTAINS conditions 43
  EXISTS conditions 44
  IN conditions 43
  IS NULL conditions 44
  LEADING SUBSTRING SEARCH conditions 40
  LIKE conditions 40
  NOT conditions 45
  subqueries 35
  three-valued logic 46
  truth value conditions 45
  SECOND function 248
  SECONDS function 248
  SELECT INTO
    Transact-SQL 626
    using REPLACE function 26, 141, 189, 195, 196, 200, 201, 241, 243, 244, 245, 248, 268, 329
  SELECT statement
    examples 527
    Transact-SQL 617
  selectivity
    explicit 47
    hints 48
    user-supplied conditions 47
  server
    properties 121
  server administration
    SQL Anywhere and IQ 635
  services
    registry entries 14
  SET OPTION statement
    Transact-SQL 616
  SHARED syntax
    Transact-SQL 621
  SIGN function 249
  SIGNAL statement
    Transact-SQL 633
  SIMILAR function 250
  SIN function 250
  SMALLDATETIME data type 83
  SMALLINT data type 73
  SMALLMONEY data type 76
  SOME conditions 39
  SORTKEY function 251
  SOUNDEX function 255
  sp_expireallpasswords system procedure 354
  sp_iq_reset_identity system procedure 454
  sp_iqaddlogin system procedure 355
Index

sp_iqbrestoreaction stored procedure 454
sp_iqcheckdb
  allocation mode 361
  check mode 362
  DBCC_LOG_PROGRESS option 365
  dropleaks mode 363
  output 365
  performance 364
  resetclocks option 362
  sample output 365
  syntax 359
  time to run 364
  verify mode 362
sp_iqcheckdb system procedure 359
sp_iqcheckoptions system procedure 366
sp_iqcolumn system procedure 370
sp_iqcolumnuse system procedure 372
sp_iqconnection system procedure 373
sp_iqcontext system procedure 378
sp_iqcopyloginpolicy system procedure 380, 437
sp_iqcursorinfo system procedure 381
sp_iqdatatype system procedure 384
sp_iqdbsize system procedure 386
sp_iqdbspace system procedure 388
sp_iqdbspaceinfo system procedure 391
sp_iqdbspaceobjectinfo system procedure 394
sp_iqdbstatistics system procedure 397
sp_iqdroplogin system procedure 399
sp_iqemptyfile system procedure 399
sp_iqestdbspaces system procedure 402
sp_iqestjoin system procedure 400
sp_iqestspace system procedure 404
sp_iqevent system procedure 404
sp_iqfile system procedure 407
sp_iqhelp system procedure 408
sp_iqindex system procedure 415
sp_iqindex_alt system procedure 415
sp_iqindexadvice system procedure 418
sp_iqindexfragmentation system procedure 419
sp_iqindexinfo
  displaying index information 423, 424
sp_iqindexinfo system procedure 421
sp_iqindexmetadata system procedure 423
sp_iqindexsize system procedure 425
sp_iqindexuse system procedure 426
sp_iqjoinindex system procedure 427
sp_iqjoinindexsize system procedure 431
sp_iqlocks system procedure 434
sp_iqmodifylogin 438
sp_iqmodifylogin system procedure 438
sp_iqobjectinfo system procedure 438
sp_iqpassword system procedure 441
sp_iqprimary system procedure 442
sp_iqprocedure system procedure 443
sp_iqprocparm system procedure 446
sp_iqrebuildindex system procedure 450, 456
sp_iqrename system procedure 452
sp_iqsetcompression system procedure 352
sp_iqshowcompression system procedure 352
sp_iqshowpsxe system procedure 457
sp_iqspaceinfo system procedure 459
  sample output 460
sp_iqspaceused system procedure 460
sp_iqstatistics system procedure 461
sp_iqstatus system procedure 463
  sample output 464
sp_iqsysmon system procedure 466
sp_iqtable system procedure 472
sp_iqtablesizer system procedure 475
sp_iqtableuse system procedure 477
sp_iqtransaction system procedure 477
sp_iqunusedcolumn system procedure 481
sp_iqunusedindex system procedure 482
sp_iqunusedtable system procedure 483
sp_iqversioninfo system procedure 484
sp_iqview system procedure 486
sp_iqwho system procedure 487
sp_iqworkmon system procedure 490
sp_login_environment system procedure 513
sp_remote_columns system procedure 513
sp_remote_exported_keys system procedure 514, 515
sp_remote_primary_keys system procedure
  syntax 516
sp_remote_tables system procedure 517
sp_servercaps system procedure 518
sp_tsql_environment system procedure 520
SPACE function 256
special characters
  in strings 22
special values
<table>
<thead>
<tr>
<th>SQL function</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT DATABASE special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT DATE special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT PUBLISHER special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT TIME special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT TIMESTAMP special value</td>
<td>54</td>
</tr>
<tr>
<td>CURRENT USER special value</td>
<td>54</td>
</tr>
<tr>
<td>LAST USER special value</td>
<td>54</td>
</tr>
<tr>
<td>SQLCODE special value</td>
<td>55</td>
</tr>
<tr>
<td>SQLSTATE special value</td>
<td>55</td>
</tr>
<tr>
<td>TIMESTAMP special value</td>
<td>56</td>
</tr>
<tr>
<td>USER special value</td>
<td>56</td>
</tr>
<tr>
<td>SQL IX dialect differences</td>
<td>343</td>
</tr>
<tr>
<td>user-defined functions</td>
<td>123</td>
</tr>
<tr>
<td>SQL Anywhere administrative roles</td>
<td>596</td>
</tr>
<tr>
<td>documentation xo</td>
<td>xx</td>
</tr>
<tr>
<td>referential integrity constraints</td>
<td>610</td>
</tr>
<tr>
<td>SQL functions compatibility</td>
<td>622</td>
</tr>
<tr>
<td>ERRORMSG function syntax</td>
<td>166</td>
</tr>
<tr>
<td>GRAPHICAL_PLAN function syntax</td>
<td>174</td>
</tr>
<tr>
<td>GROUP_MEMBER function syntax</td>
<td>177</td>
</tr>
<tr>
<td>GROUPING function syntax</td>
<td>177</td>
</tr>
<tr>
<td>HTML_PLAN function syntax</td>
<td>183</td>
</tr>
<tr>
<td>ISDATE function syntax</td>
<td>191</td>
</tr>
<tr>
<td>ISNUMERIC function syntax</td>
<td>192</td>
</tr>
<tr>
<td>LEN function syntax</td>
<td>196</td>
</tr>
<tr>
<td>NEWID function syntax</td>
<td>208</td>
</tr>
<tr>
<td>REVERSE function syntax</td>
<td>244</td>
</tr>
<tr>
<td>STR_REPLACE function syntax</td>
<td>261</td>
</tr>
<tr>
<td>STRTOUUID function syntax</td>
<td>263</td>
</tr>
<tr>
<td>UUIDTOSTR function syntax</td>
<td>331</td>
</tr>
<tr>
<td>SQL syntax</td>
<td></td>
</tr>
<tr>
<td>CURRENT DATABASE special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT PUBLISHER special value</td>
<td>53</td>
</tr>
<tr>
<td>CURRENT USER special value</td>
<td>54</td>
</tr>
<tr>
<td>LAST USER special value</td>
<td>54</td>
</tr>
<tr>
<td>TIMESTAMP special value</td>
<td>56</td>
</tr>
<tr>
<td>USER special value</td>
<td>56</td>
</tr>
<tr>
<td>SQL92 conformance</td>
<td>343</td>
</tr>
<tr>
<td>SQLCODE special value</td>
<td>55</td>
</tr>
<tr>
<td>SQLCONNECT environment variable</td>
<td>12</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td></td>
</tr>
</tbody>
</table>
Index

insert 189
length 133, 136
position 136
STRING function 263
string functions 115
ASCII 129
BIT_LENGTH 132
BYTE_LENGTH 133
CHAR 135
CHAR_LENGTH 136
CHARINDEX 136
DIFFERENCE 165
INSERTSTR 189
LCASE 194
LEFT 195
LENGTH 197
LOCATE 198
LOWER 200
LTRIM 201
OCTET_LENGTH 216
PATINDEX 217
REPEAT 241
REPLACE 242
REPLICATE 243
RIGHT 245
RTRIM 247
SIMILAR 250
SORTKEY 251
SOUNDEX 255
SPACE 256
STR 260
STRING 263
STUFF 264
SUBSTR 265
SUBSTRING 265
TRIM 268
UCASE 329
UPPER 329
strings
about 22
catenating 26, 243, 263
catenation operators 26
constants 22, 25
converting to lowercase 194, 200
converting to uppercase 329
delimiter 31
determining length 197
determining similarity 250
literal strings 22
removing blanks 268
removing leading blanks 201
removing trailing blanks 247
replacing substrings 242
returning a substring 265
SOUNDEX function 255
special characters 22
Transact-SQL 31
STRTOUUID function
SQL syntax 263
STUFF function 264
subqueries
    Adaptive Server Enterprise 618
disjunction of 36
    in expressions 25
    in search conditions 35
    IQ 618
    IQ implementation 345
    SQL Anywhere 618
subqueries, disjunction of 37
SUBSTR function 265
SUBSTRING function 265
SUM function 265
summary 356
USER_ID function 266
USER_NAME function 266
SYBASE environment variable 12
Sybase IQ User Administration
    sp_iqdroplogin 399
SYBASE_JRE environment variable 13
SYBASE_JRE environment variable 13
SQL identifiers 21
TIMESTAMP special value 56
USER special value 56
SYSIQBACKUPHISTORY system view 546
SYSIQBACKUPHISTORYDETAIL system view 548
SYSIQDBFILE system view 549
Index

SYSI奎DBSPACE system view 550
SYSI奎IDX system view 551
SYSI奎JOINIDX system view 554
SYSI奎JOINIXCOLUMN system view 555
SYSI奎JOINIXTABLE system view 556
SYSI奎PARTITIONCOLUMNS system view 557
SYSI奎TAB system view 558
SYSI奎PARTITION system view 566
SYSI奎PARTITIONKEY system view 566
SYSI奎HELPSCHEME system view 567
SYSI奎SUBPARTITIONKEY system view 575

system administrator
    Adaptive Server Enterprise 600

system calls
    from stored procedures 525
    xp_cmdshell system procedure 525

system catalog 534
    Adaptive Server Enterprise compatibility 599
    Transact-SQL 587

system functions 118
    COL_LENGTH 138
    COL_NAME 138
    CONNECTION_PROPERTY 139
    DATALength 148
    DB_ID 161
    DB_NAME 162
    DB_PROPERTY 162
    EVENT_CONDITION 166
    EVENT_CONDITION_NAME 168
    EVENT_PARAMETER 168
    INDEX_COL 188
    NEXT_CONNECTION 209
    NEXT_DATABASE 209
    OBJECT_ID 215
    OBJECT_NAME 216
    PROPERTY 225
    PROPERTY_DESCRIPTION 225
    PROPERTY_NAME 226
    PROPERTY_NUMBER 227
    SUSER_ID 266
    SUSER_NAME 266
    Transact-SQL 624
    USER_ID 330
    USER_NAME 330

system procedures
    about 351
displaying information about 408
    sa_audit_string 492
    sa_checkpoint_execute 492
    sa_conn_activity 493
    sa_conn_info 494
    sa_conn_properties 495
    sa_db_info 496
    sa_db_properties 497
    sa_dependent_views 353
    sa_disable_auditing_type 500
    sa_enable_auditing_type 498
    sa_eng_properties 498
    sa_flush_cache 501
    sa_make_object 501
    sa_rowgenerator 502
    sa_server_option 504
    sa_set_http_header 510
    sa_set_http_option 511
    sa_table_page_usage 500
    sa_validate 511
    sa_verify_password 354, 512
    sp_expireallpasswords 354
    sp_iqaddlogin 355
    sp_iqbackupdetails 356
    sp_iqbackupsummary 358
    sp_iqcheckdb 359
    sp_iqcheckoptions 366
    sp_iqlclient_lookup 368
    sp_iqcolumn 370
    sp_iqclientmuse 372
    sp_iqconnection 373
    sp_iqcontext 378
    sp_iqcopyloginpolicy 380, 437
    sp_iqcursorinfo 381
    sp_iqdatatime 384
    sp_iqdbsize 386
    sp_iqdbspaceobjectinfo 394
    sp_iqdbstatistics 397
    sp_iqdroplogin 399
    sp_iqemptyfile 399
    sp_iqestdbspaces 402
    sp_iqestjoin 400
    sp_iqestspace 404
    sp_iqevent 404
    sp_iqfree 407
    sp_iqhelp 408

Reference: Building Blocks, Tables, and Procedures 659
Index

sp_iqindex 415
sp_iqindex_alt 415
sp_iqindexadvice 418
sp_iqindexesize 425
sp_iqindexuse 426
sp_iqjoindex 427
sp_iqjoindexsize 431
sp_iqmodifylogin 438
sp_iqobjectinfo 438
sp_iqpassword 441
sp_iqprocedure 443
sp_iqprocparm 446
sp_iqrename 452
sp_iqrestoreaction 454
sp_iqsetcompression 352
sp_iqshowcompression 352
sp_iqshowspexe 457
sp_iqspaceinfo 459
sp_iqspaceused 460
sp_iqstatistics 461
sp_iqstatus 463
sp_iqtable 472
sp_iqtablesize 475
sp_iqtableuse 477
sp_iqtransaction 477
sp_iqunusedcolumn 481
sp_iqunusedindex 482
sp_iqunusedetable 483
sp_iqversionuse 484
sp_iqview 486
sp_iqwho 487
sp_iqworkmon 490
sp_login_environment 513
sp_remote_columns 513
sp_remote_exported_keys 514, 515
sp_remote_primary_keys 516
sp_remote_tables 517
sp_servercaps 518
sp_tsq1_environment 520
xp_cmdshell 525

System security officer
  Adaptive Server Enterprise 600

System tables
  about 591

Adaptive Server Enterprise compatibility 599
displaying information 408
DUMMY 594
system variables 59
system views
  Adaptive Server Enterprise 639
  consolidated 527
SYSARTICLE 532
SYSARTICLECOL 532
SYSARTICLECOLS 532
SYSARTICLES 532
SYSCAPABILITIES 533
SYSCAPABILITY 533
SYSCAPABILITYNAME 533
SYSCATALOG 534
SYSCHECK 534
SYSCOLAUTH 534
SYSCOLPERM 535
SYSCOLSTAT 536
SYSCOLSTATS 536
SYSCOLUMN 537
SYSCOLUMNM 537
SYSCOLUMNMS 537
SYSCOLUMNMS ASE compatibility view 537
SYSCONSTRAINT 538
SYSDBF 538
SYSDBSPACE 539
SYSDBSPACEPERM 539
SYSDependency 539
SYSDomain 540
SYSEVENT 540
SYSEVENTTYPE 540
SYSEXternalLOGIN 541
SYSEXTERNENV 541
SYSEXTERNENVOBJECT 541
SYFILE 542
SYFKCOL 542
SYFKKEY 542
SYSFOREIGNKEY 543
SYSFOREIGNKEYS 543
SYSGROUP 543
SYSGROUPS 544
SYSHISTORY 544
SYSID 545
SYSIDCOL 545
Index

SYSINDEX 545
SYSINDEXES 546
SYSINDEXES ASE compatibility view 546
SYSIINFO 546
SYSIQBACKUPHISTORY 546
SYSIQBACKUPHISTORYDETAIL 548
SYSIQCOLUMN 549
SYSIQDBFILE 549
SYSIQDBSPACE 550
SYSIQFILE 551
SYSIQIDX 551
SYSIQINFO 552
SYSIQTAB 558
SYSIQJOINIDX 554
SYSIQJOININDEX 555
SYSIQJOINXCOLUMN 555
SYSIQJOINXTABLE 556
SYSIQMPXLOGINPOLICYOPTION 557
SYSIQMPXSERVER 557
SYSIQOBJECTS ASE compatibility view 557
SYSIQPARTITIONCOLUMN 557
SYSIQTABCOL 559
SYSIQTABLE system view 561
SYSIQVINDEX ASE compatibility view 561
SYSIQCOL compatibility view 561
SYSIJAR 561
SYSIARCOMPONENT 562
SYSIJAVACLASS 562
SYSILOGINMAP 562
SYSILOGINPOLICY 563
SYSILOGINPOLICYOPTION 563
SYSILOGINS ASE compatibility view 563
SYSIMVOPTION 563
SYSIOBJECTS ASE compatibility view 564
SYSIOPTIONS 565
SYSIPARTITION 566
SYSIPARTITIONKEY 566
SYSIPARTITIONSCEME 567
SYSIPIHSIDX 568
SYSIPROCAUTH 568
SYSIPROCEDURE 568
SYSIPROCERMS 569
SYSIPROCPerm 569
SYSIPROC 570
SYSIROXYTAB 570
SYSPPUBLICATION 570
SYSPUBLICATIONS 571
SYSRMARK 571
SYSRREMOTE OPTION 571
SYSRREMOTE OPTION2 572
SYSRREMOTE OPTIONS 572
SYSRREMOTE OPTIONTYPE 572
SYSRMOTETYPE 573
SYSRMOTYPE 573
SYSRMOTYPE 573
SYSRMOTYPEUERS 574
SYSSCHEDULE 574
SYSSOURCE 575
SYSSQLSERVERTYPE 575
SYSSUBPARTITIONKEY 575
SYSSUBSCRIPTION 575
SYSSUBSCRIPTIONS 576
SYSSYNC 576
SYSSYNCPROFILE 577
SYSSYNSCS 577
SYSSYNCSUBSCRIPTIONS 578
SYSSYNSCUERS 578
SYSTAB 579
SYSTABAUTH 579
SYSTABCOL 580
SYSTABLE 579
SYSTABLEPERM 580
SYTTEXTCONF 580
SYTTEXTIDX 581
SYTTEXTIDXTAB 581
SYTHRIGGER 581
SYSTHRIGGERS 582
SYSTYPEMAP 582
SYSTYPES ASE compatibility view 582
SYTUSER 583
SYTUSERAUTH 583
SYTUSERAUTHORITY 583
SYTUSERLIST 584
SYTUSERMESSAGE 584
SYTUSEROPTIONS 584
SYTUSERPERM 585
SYTUSERPERMS 585
SYTUSERERS ASE compatibility view 586
SYTUSERETYPE 585
SYTVIEW 586
SYTVIEWS 586
SYTWEBSERVICE 586
Index

T

tables
  displaying information about 408
  iq_dummy 102
  number per join index 349
  Transact-SQL 609
TAN function 267
tangent 267
temp_cache_memory_mb
  sa_server_option parameter 510
temporary tables
  Transact-SQL 612
TEXT data type 70, 602
  compatibility 602, 606
THEN
  IF expression 29
threads
dbcc 361
three-valued logic
  about 46
  NULL value 67
TIME data type 83
Time Series functions 124
  error handling 125
  error logging 126
  IMSL library 124
time series functions 123
times
  queries 85
TIMESTAMP
  converting an expression 159
  data type 83
  data type compatibility 604, 605
  special value 56
TINYINT data type 73
TODAY function 267, 594
trailing blanks
trimming 71, 602
transaction log
adding string 492
transaction management
  monitoring with sp_iqsysmon 466
Transact-SQL
  about 595
  batches 627
  bitwise operators 27
  comparison conditions 35
  constants 31
  creating compatible databases 607
  expressions 31
  joins 620
  local variables 58
  outer join operators 28
  overview 596
  procedure language overview 626
  procedures 627
  referential integrity constraints 610
  result sets 630
  strings 31
  system catalog 587
  user-defined data types 89
  variables 631
  writing portable SQL 617
Transact-SQL compatibility
databases 607
triggers
  not supported 612
TRIM function 268
  trimming trailing blanks 71, 602
troubleshooting
  logging operations 508
  request_level_logging 504
TRUNCNUM function 268
TS_ARMA_AR function 269
TS_ARMA_CONST function 272
TS_ARMA_MA function 276
TS_AUTO_UNI_AR function 283
TS_AUTOCORRELATION function 280
TS_BOX_COX_XFORM function 287
TS_DIFFERENCE function 289
TS_ESTIMATE_MISSING function 294
TS_LACK_OF_FIT function 297
TS_LACK_OF_FIT_P function 301
TS_MAX_ARMA_AR function 304
TS_MAX_ARMA_CONST function 308
TS_MAX_ARMA_LIKELIHOOD function 311
TS_MAX_ARMA_MA function 315
TS_OUTLIER_IDENTIFICATION function 318
TS_PARTIAL_AUTOCORRELATION function 323
TS_VWAP function 327
type conversions 89
types

662

Sybase IQ
Index

about data types 69

U

UCASE function 329
UNION
in subqueries 618
UNIQUEIDENTIFIER data type 81
UNIQUEIDENTIFIERSTR data type about 69
universally unique identifiers
SQL syntax for NEWID function 208
UPPER function 329
usefulness hints 51
USER
special constant 594
special value 56
user administration. see login management
user IDs
Adaptive Server Enterprise 614
case sensitivity 608
determining from user name 266, 330
user name
determining from user ID 266, 330
USER_ID function 330
USER_NAME function 330
user-defined data types
about 87
case-sensitivity 608
Transact-SQL 89
user-defined functions 122
compatibility 625
users
adding 355
displaying information about 487
dropping 399
modifying 438
number of connections 373
user-supplied condition hint strings 48
user-supplied condition hints, guidelines and usage 52
user-supplied condition selectivity 47
user-supplied conditions
for queries 46, 47
user-supplied hints on join equality conditions 51
utilities
SQL Anywhere 635
UUIDs
SQL syntax for NEWID function 208
SQL syntax for STRTOUUID function 263
SQL syntax for UUIDTOSTR function 331
UUIDTOSTR function
SQL syntax 331

V
validating
catalog store 511
VAR_POP function 331
VAR_SAMP function 332
VARBINARY data type 77
VARCHAR data type
about 69, 70
trimming trailing blanks 71, 602
variables
about 56
connection-level 59
global 57, 59
local 57
Transact-SQL 631
VARIANCE function 333
verifying
indexes 362
keys 362
partitioned tables 363
passwords 354, 512
views
displaying information about 408
system views 586
updatable 626

W
WEEKS function 335
WEIGHTED_AVG function 336
WHERE clause
Transact-SQL 619
WIDTH_BUCKET function 337
window functions
Index

window function type 105
window name or specification 105
window partition 105
window functions, defining 105
window name 105
window specification 105
window type 105
windows aggregate functions 104
WITHIN GROUP clause 106

X
xp_cmdshell system procedure
   syntax 525

Y
YEAR function 339
YEARS function 339
YMD function 341

Z
zero-length strings
   Transact-SQL 621